



Measuring, Understanding, and Improving Content Distribution Technologies

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor Ingenieur (Dr.-Ing.)

vorgelegt von
Hani Salah, M.Sc.
geboren in Hebron, Palästina

Tag der Einreichung: 01.10.2015

Tag der Disputation: 12.11.2015

Referenten: Prof. Dr. Thorsten Strufe, TU Dresden
Prof. Dr. Max Mühlhäuser, TU Darmstadt

Darmstadt 2016
Darmstädter Dissertationen
D17

Ehrenwörtliche Erklärung ¹

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades “Doctor Ingenieur (Dr.-Ing.)” mit dem Titel “Measuring, Understanding, and Improving Content Distribution Technologies” selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 01.10.2015

Hani Salah, M.Sc.

¹Gemäß §9 Abs. 1 der Promotionsordnung der TU Darmstadt

Acknowledgements

Looking back over the past four years, I have realized it would not have been possible to complete this work without the support of several people. Hence, I would like to express my appreciation to them.

First and foremost, I would like to express my farthest gratitude and deep appreciation to my advisor Prof. Dr. Thorsten Strufe for providing me with time and continuous support through indispensable advice and expert opinion. His supervision is really splendid and was essential for me to reach this stage. I would like also to express my deep thanks to Prof. Dr. Max Mühlhäuser for accepting to review my dissertation. It is an honour to have his name on my dissertation.

I am very thankful to the German Academic Exchange Service (DAAD) for financing my study in Germany. I am also grateful to Stefanie Roos for the very fruitful cooperation, and to Thomas Paul for the valuable discussions and for translating the abstract of this dissertation to German. Special thanks go to Benjamin Schiller, Giang Nguyen, Abdalrahman Eweiwi, and Tim Grube for the very useful comments that improved the quality of this work.

Special thanks go to my colleagues in the P2P, Telecooperation, and Knowledge Engineering research groups for creating a friendly work environment during my stay at TU Darmstadt. I would like also to thank Julian Wulfheide, Sven Frese, and Oleksii Donets for their help with simulations and measurements.

I would like to take a moment to thank my uncles Dr. Adnan Salah (God bless his soul) and Mr. Ziad Salah for their encouragement. I also thank Dr. Hashem Tamimi and Prof. Karim Tahboub both for their valuable advices and for supporting my scholarship application. Sincere thanks go to Ahmed Fahmy Yousef, Ghannam Aljabari, Hisham Ihshaish, Majed Shaheen, Mohammed Srour, Mohamed Saied Emam, and Ziad Ghaith who have been extremely supportive through their personal drive to push me towards completing the PhD. I am grateful to have had the opportunity to call them true friends.

Last but not least, my thanks and appreciations go to my parents Sameeh and Hayat, my wife Ashwaq and our two little kids Sameeh and Abdulrahman for their continuous support and love.

Dedicated to my parents, to my wife and our kids.

Abstract

The Internet plays a focal role in our social life, work, education, and entertainment. The current Internet has been designed, over 50 years ago, for reliable host-to-host communications. Today, however, the Internet is mainly used for content distribution and retrieval.

To cope with this shift in the Internet usage, three technologies for efficient content distribution were developed in the past years: Peer-to-Peer (P2P) systems, Content Delivery Networks (CDNs), and Information-Centric Networking (ICN). P2P systems and CDNs are widely used today, both constructing overlays atop the current Internet. ICN, in contrast, is a promising research direction proposing to redesign the Internet from scratch as a content-centric network.

We focus on P2P systems and ICN. Despite the wide deployment of P2P systems and the promising features of ICN, several research problems in these technologies are still awaiting for effective solutions. We address in this dissertation three of them.

We first deal with Kademlia, the most popular P2P system. Improving and giving guarantees on Kademlia's performance and robustness is needed to achieve a reliable and high quality service. This requires accurate understanding of system-wide topological properties such as the degree distribution and graph resilience, as well as properties of routing information. In spite of the importance of these properties, they were analysed so far only through partial measurements or oversimplified simulations. Instead, accurate results can be derived only from graph snapshots of real systems. However, capturing such snapshots is challenging since it requires to collect large amount of highly dynamic information, almost instantly, which was not provided by previous measurements. Results of such graph snapshots also can be used to develop accurate simulation models.

We address this challenge by KadSpider, a crawler that we develop to capture highly representative graph snapshots of KAD (a popular implementation of Kademlia). Exploiting KAD design, KadSpider downloads the target information several times faster than prior Kademlia crawlers with lower signalling overhead.

Analysing the captured snapshots and comparing them to synthetic graphs generated by simulations, we provide important information about KAD. We also show that, for certain properties, simulative results can significantly deviate from the results of the real system. Most interestingly, the complete graph in the real system, due to a greatly increased ratio of stale routing information, is much more vulnerable to targeted attacks than in simulations. This observation holds even when we simulate with a widely accepted churn model.

Using the captured graph snapshots, we analyse the diversity of node’s neighbours over the corresponding identifier space sections. We also show that the average routing hop count can be reduced when this diversity is maximized. Consequently, we apply this strategy in a modified neighbour selection scheme. The scheme is backward compatible with Kademlia and its derivatives, and incurs only negligible computational overhead. We demonstrate the utility of the scheme via theoretical model derivations, simulations of three notable Kademlia-type systems, and measurements on KAD nodes.

After that, we shift our attention to Named-Data Networking (NDN), widely considered a promising ICN architecture for the future Internet. In particular, we address two salient problems in NDN: (i) cache management and (ii) defending against an NDN-tailored DDoS attack called the Interest Flooding Attack (IFA).

We argue that both problems should be addressed in a coordinated way based on timely and network-wide content access information and other system states, which was not achieved in the past. The distributed nature of the Internet, in addition to the huge volumes and high dynamics of coordination information, however, make realizing such coordination challenging.

We address this challenge by CoMon, a novel framework for **C**oordination in NDN that is based on (i) lightweight **M**onitoring of content access and resource usage and (ii) bounded advertisement of relevant information. CoMon assigns monitoring tasks to a small number of monitoring nodes selected such that the entire network traffic passes, or is enforced to pass, through them at an early stage. The monitoring nodes, in cooperation with a centralized controller, are also responsible for re-routing user requests towards in-network caches as well as for defending against IFAs. Our evaluations, based on extensive simulations, show that CoMon incurs negligible signalling overhead. They also show that CoMon enables to achieve both a remarkable caching efficiency improvement and high robustness against IFAs, in comparison both to the original system as well as to the state of the art.

Zusammenfassung

Das Internet spielt eine wichtige Rolle im Leben seiner Nutzer. Man verwendet es für gesellschaftliche Zwecke ebenso wie für Arbeit, Bildung und Unterhaltung. Das derzeitige Internet wurde bereits vor 50 Jahren entworfen um verlässliche Verbindungen zwischen Rechnern zu schaffen. Die heutige Verwendung beinhaltet jedoch hauptsächlich das Verteilen von medialen Inhalten.

Um dieser Veränderung der Nutzung des Internet Rechnung zu tragen wurden in den letzten Jahren drei verschiedene Technologien entwickelt: Peer-to-Peer (P2P) Systeme, Content Delivery Networks (CDNs) und Information-Centric-Networking (ICN). P2P und CDN Technologien werden heutzutage intensiv genutzt. ICN Technologie ist demgegenüber ein vielversprechendes Forschungsfeld, mit dem Ziel das Internet von Grund auf neu zu entwerfen.

Der Fokus dieser Dissertation liegt auf P2P und ICN Systemen. Trotz der großflächigen Nutzung von P2P und den vielversprechenden Eigenschaften von ICN Systemen müssen noch immer diverse Forschungsfragen effizient gelöst werden. Ziel ist daher die Verbesserung von P2P Systemen und das Erbringen von Beiträgen zur Realisierung von ICNs.

Zuerst wird dazu das am weitesten verbreitete P2P System, KAD (eine populäre Implementierung von Kademlia), hinsichtlich der Eigenschaften von Topologie und Routing-Tabellen untersucht. Im Gegensatz zu vorhandenen Arbeiten werden Momentaufnahmen von Graphen realer Systeme verwendet. Das Erstellen dieser Momentaufnahmen erfordert jedoch eine neue Methode, da existierende Verfahren nicht in der Lage sind große Mengen sich schnell verändernder Information nahezu verzögerungsfrei zu speichern. Zu diesem Zweck wurde der KadSpider entwickelt. Dieser nutzt das Protokolldesign von KAD um repräsentative Momentaufnahmen um ein Vielfaches schneller und mit weniger Kommunikationsaufwand als bekannte Crawler zu erstellen.

Die Analyse der Momentaufnahmen liefert wichtige Informationen über KAD. Insbesondere wird in dieser Arbeit gezeigt, dass Simulationen, welche die Basis der bisherigen Forschung bilden, von realen Systemen in bestimmten Eigenschaften abweichen. Am interessantesten ist hierbei, dass reale Graphen

sehr viel verwundbarer gegenüber gezielten Angriffen sind als es Simulationen vermuten lassen. Dies gilt ebenso für solche Simulationen, die mit einem weithin in der Forschung akzeptierten Churn-Modell durchgeführt wurden.

Des Weiteren werden die Momentaufnahmen in dieser Arbeit dazu verwendet die Diversität der Nachbarn in den entsprechenden Teilen der Routing-Tabelle der Knoten zu untersuchen. Außerdem wird gezeigt, dass die mittlere Entfernung (hop count) zwischen Knoten reduziert werden kann, wenn die Diversität maximiert wird. Aufgrund dieser neuen Erkenntnis wurde ein Schema für die Nachbarschaftswahl in KAD entwickelt. Dieses ist abwärtskompatibel mit Kademia und dessen Derivaten und erzeugt nur vernachlässigbaren Rechenaufwand. Die Nützlichkeit des neuen Schemas wird durch theoretische Überlegungen, Simulationen in drei Kademia-ähnlichen Systemen und durch Messungen auf KAD Knoten nachgewiesen.

Im Anschluß daran werden in dieser Arbeit zwei wichtige Probleme der NDN Architektur, einer vielversprechenden ICN Architektur, betrachtet. Das ist zum einen das (i) Zwischenspeicher-Management und zum anderen (ii) die Verteidigung gegen Interest Flooding Attacks (IFA), welche die Eigenheiten der NDN Architektur ausnutzen. Im Rahmen dieser Arbeit wird gezeigt, wie beide Probleme mithilfe netzwerkweiter Zugriffs- und Systemstatusinformationen gelöst werden können. Die dezentrale Architektur des Internets, die große Dynamik und das hohe Volumen an Statusinformationen stellen eine große Herausforderung in Hinblick auf die Implementierung dar.

Um dieser Herausforderung zu begegnen ist CoMon, ein neuartiges Framework, entwickelt worden. Es bietet (i) die Überwachung von Zugriffen auf Inhalte und Ressourcen und (ii) eine begrenzte Verbreitung von relevanten Statusinformationen. CoMon verteilt dafür Überwachungsaufträge an eine kleine Anzahl an Knoten, die so selektiert werden, dass an ihnen der gesamte Netzwerkverkehr anliegt. Diese Netzwerkknoten sind, in Verbindung mit einem zentralen Controller, für das Weiterleiten von Anfragen an netzwerkinterne Zwischenspeicher und die Verteidigung gegen IFAs verantwortlich. Die Evaluationen, basierend auf Simulationen, zeigen, dass CoMon vernachlässigbaren Kommunikationsaufwand erzeugt. Sie zeigen außerdem, dass CoMon verglichen mit der originalen Variante und dem Stand der Technik sowohl eine Steigerung der Effizienz der Zwischenspeicher, als auch eine erhöhte Robustheit gegenüber IFAs erzielt.

Contents

Abstract	vii
Zusammenfassung	ix
List of Figures	xvi
List of Tables	xviii
1 Introduction	1
1.1 Problem Statements	3
1.1.1 Characterizing Topological and Routing Table Properties of Kademlia	3
1.1.2 Coordinating Cache Management in NDN	4
1.1.3 Defending Against DDoS Attacks in NDN	4
1.2 Research Methodology	4
1.3 Contributions	5
1.4 Organization	6
1.5 Use of Published Work	7
I Understanding and Improving Kademlia	9
2 Overview of Kademlia-type Systems	11
2.1 The Original Kademlia	11
2.2 MDHT and iMDHT	13
2.3 KAD	13
2.3.1 Routing Table Maintenance	14
2.3.2 Bootstrapping	14
2.3.3 Populating the Routing Table	15
2.3.4 Protection Against Flooding Attacks	15
2.4 Summary	15
3 Capturing Accurate Snapshots of the KAD Graph	17
3.1 Related Work	18

3.2	Crawling Task and Challenges	19
3.3	Design and Implementation	19
3.3.1	Advanced Crawling Techniques	19
3.3.2	Algorithmic Design	23
3.4	Dataset and Performance Results	25
3.4.1	Measurement Setup and Dataset Description	25
3.4.2	Crawling Performance	27
3.5	Discussion	30
3.6	Summary	31
4	Topological and Routing Table Properties of KAD	33
4.1	Related Work	34
4.2	Background	35
4.2.1	Topological and Routing Table Properties	35
4.2.2	Standard Graph Models	37
4.3	Methodology	38
4.3.1	Measurements	39
4.3.2	Simulations	39
4.4	Results	40
4.4.1	Degree Distribution	40
4.4.2	Routing Table Completeness	44
4.4.3	Graph Resilience	47
4.5	Summary	50
5	Improving the Routing Performance in Kademlia-type Systems	53
5.1	Related Work	54
5.2	Improving Routing Performance	55
5.2.1	The Idea	55
5.2.2	Diversity Degrees in a Real Kademlia-type System	57
5.2.3	Implementation	57
5.3	Evaluation	58
5.3.1	Performance Gain: Full Deployment	58
5.3.2	Performance Gain: Partial Deployment in KAD	61
5.3.3	Impact of Churn and System Size	63
5.4	Summary	66
II	Improving Named-Data Networking	67
6	Overview of Named-Data Networking	69
6.1	Overview of NDN	69
6.2	Operation Primitives and Packet Types	70

6.3	Content Naming	70
6.4	In-network Caching	71
6.5	Node Model	71
6.6	Packet Handling	72
6.7	Content-based Security	72
6.8	Summary	73
7	Coordinated Cache Management in NDN	75
7.1	Motivating Example	76
7.2	Related Work	78
7.3	Design Requirements and Overview	80
7.3.1	Design Requirements	81
7.3.2	Architecture and Operation Primitives	81
7.4	Design Specifications	83
7.4.1	Cache-aware Routing	83
7.4.2	Caching-related Decisions	84
7.4.3	Selection of Monitoring Nodes	85
7.4.4	Reducing the Frequency of Updates	89
7.5	Evaluation	90
7.5.1	PRCS: Coverage and Closeness to Sources	91
7.5.2	CoMon: Caching Efficiency Gain and Overhead	93
7.6	Summary	100
8	Coordinated Defence Against Interest Flooding in NDN	101
8.1	Interest Flooding Attack	102
8.2	Related Work	104
8.3	Design Requirements and Overview	105
8.3.1	Design Requirements	105
8.3.2	Architecture and Operation Primitives	105
8.4	Design Specifications	107
8.4.1	Detection of IFAs	107
8.4.2	Reaction Against Potential IFAs	108
8.5	Evaluation	109
8.5.1	Simulation Setup	109
8.5.2	Evaluation Metrics	110
8.5.3	Results	111
8.6	Summary	116
9	Summary and Outlook	117
9.1	Summary	117
9.1.1	Understanding and Improving Kademlia	117
9.1.2	Improving Named-Data Networking	119

9.2 Outlook	120
Bibliography	123
A Search Keywords Used in KAD Measurements	135
B Acronyms	139
C Author's Publications	141
D Wissenschaftlicher Werdegang des Verfassers	143

List of Figures

1.1	Forecasts for the monthly Internet traffic from 2013 to 2018	2
1.2	Research methodology	5
2.1	Routing table structures of three Kademlia-type systems	13
3.1	Exemplary sub-trees in a KAD routing table	21
3.2	Crawling architecture of KadSpider	22
3.3	Geographical distribution of KAD nodes	27
3.4	Stability ratios of online KAD nodes and routing tables	29
4.1	Four graphs extracted from a snapshot of KAD graph	38
4.2	Degree distributions of measured KAD graphs	42
4.3	Degree distributions of complete KAD graphs	43
4.4	Degree distributions of active KAD graphs	44
4.5	Actual routing table completeness	46
4.6	Distribution of missing routing table entries	47
5.1	Exemplary MDHT routing table	56
5.2	Diversity degrees of standard KAD buckets	58
5.3	Hop count distributions: model derivations vs. simulations	60
5.4	Hop count distribution: measurements on KAD nodes	63
5.5	Improvement achieved by the modified scheme: simulations of three Kademlia-type systems	64
6.1	Protocol stacks of TCP/IP and NDN	70
6.2	NDN packet types	71
6.3	Handling interest and data packets in NDN	73
7.1	Coordinated caching – a motivating example	77
7.2	System architecture of CoMon	82
7.3	The three ISP network topologies used in simulations	90
7.4	PRCS: Fraction of covered routes vs. number of monitoring nodes	91
7.5	PRCS: Hops between consumer nodes and closest monitoring node	93
7.6	Server hit ratio	95
7.7	Hop count overhead of MAR	96

7.8	Hop count overhead of FTBM	98
7.9	Signalling overhead of coordination – cache management	99
8.1	Interest flooding attack – an illustrative example	103
8.2	System architecture	106
8.3	Satisfaction ratio of legitimate interest packets	111
8.4	Global PIT usage	113
8.5	Signalling overhead of coordination – defence against IFA	115

List of Tables

3.1	Overview of KAD graph snapshots	26
3.2	Completeness statistics of KAD graph snapshots	29
3.3	KadSpider: Mapping design elements to the requirements	31
4.1	Degree analysis of KAD graphs: measured vs. synthetic graphs . .	45
4.2	Routing table completeness statistics of a measured KAD graph .	46
4.3	Resilience values of measured and synthetic KAD graphs	49
5.1	Hop counts in KAD and improvement achieved by the modified scheme	61
5.2	Hop counts, diversity degrees, and achieved improvement: measurements on KAD nodes	63
5.3	Hop counts and improvement achieved by the modified scheme: simulations of three Kademlia-type systems	65
5.4	Hop counts and improvement achieved by the modified scheme for different system sizes	66
7.1	Coordinated caching – results of the motivating example	79
7.2	The three ISP network topologies used in simulations	90
7.3	Percentage of MNs that cover all routes	92
7.4	Fraction of the closest MNs located on the first half of the route .	92
7.5	Hop count statistics of MAR	97
7.6	Hop count statistics of FTBM	99
7.7	CoMon (coordinated caching): Mapping design elements to the requirements	100
8.1	Satisfaction ratio of legitimate interest packets	112
8.2	Global PIT usage statistics	114
8.3	Signalling overhead statistics – coordinated defence against IFA .	115
8.4	CoMon (coordinated defence): Mapping design elements to the requirements	116
A.1	Search keywords used in KAD measurements	135

Introduction

The Internet has been designed in the 1960s as a network for reliable host-to-host communications. Each IP packet travels across the network from a source host to a destination host. Along its way, the packet is forwarded, hop-by-hop, towards its destination. In the past years, however, there has been a big shift in the Internet usage in contrast to what the Internet was designed for. In particular, the majority of interactions over the Internet relates today to content distribution, *e.g.* users publish content on Facebook [fac] and google+ [goo], watch and share videos on YouTube [you], and browse and share photos on Flickr [fli].

Content distribution applications generate very large and ever increasing traffic volumes, mainly due to redistribution of popular contents. Cisco forecasts [cis14] that the monthly Internet traffic will reach 132 exabytes (*i.e.* 132,000,000 terabytes) by 2018, with a 21% compound annual growth rate (see Figure 1.1). The same report states that video contents will represent 79% of the overall Internet traffic in 2018 (up from 66% in 2013).

Such massive traffic translates to high charges for network operators. In addition, with the traditional host-centric communication model, overlapping requests particularly for popular contents result in congestions and bottlenecks. This happens both in the network and at the providers' side, which lowers users' quality-of-experience (QoE). Therefore, it became doubtful that the current Internet can fulfil the demands of its users in the near future.

The increasing demand for highly scalable and efficient distribution of content over the Internet motivated the development of three technologies: (i) Content Delivery Networks (CDNs), (ii) Peer-to-Peer (P2P) systems, and (iii) Information-Centric Networking (ICN). The common idea is to request and retrieve contents by their names, irrespective of how or from where they are retrieved.

CDNs and P2P systems are popular today, both working on top of the Internet. The file-sharing P2P traffic and the traffic that crosses CDNs are expected to exceed 6,700,000 and 56,000,000 terabytes, respectively, per month by 2018 [cis14].

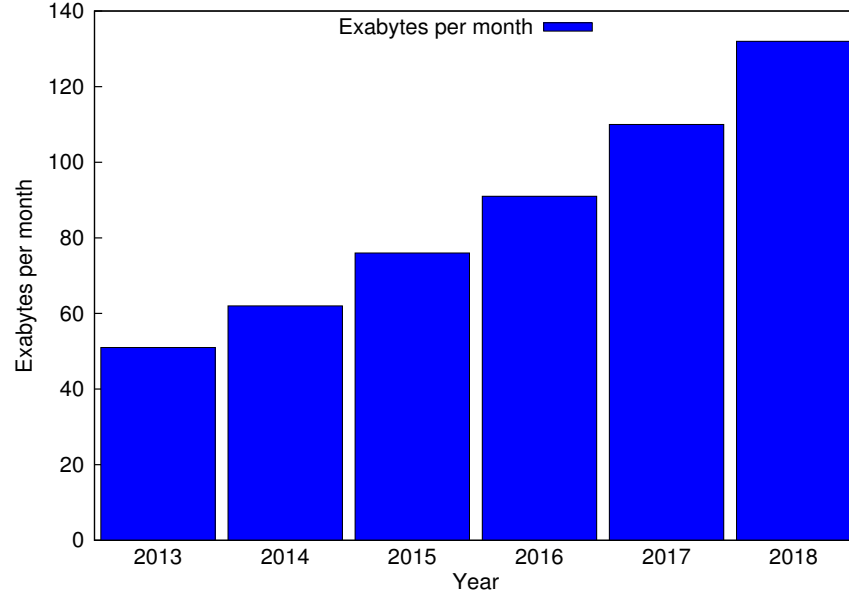


Figure 1.1: Forecasts for the monthly Internet traffic from 2013 to 2018 (adapted from [Ind14])

A CDN is composed of a number of dedicated servers, spreading across the Internet, cooperate to deliver content to users. Content providers (*e.g.* YouTube) sign up with a CDN service provider so that their contents become accessible from the CDN servers, and they pay for this service. Requests of users are then routed to the most appropriate (*e.g.* closest) server hosting a replica of the requested content. This way, CDNs decrease the workload on the origin content providers, improve the QoE, and lower the overall bandwidth consumption.

As for P2P systems, they form self-organizing overlay networks atop the Internet. The key idea is that several nodes (also called peers) share their resources among each other in a completely distributed way. More precisely, each node acts simultaneously both as a client (uses others' resources) and as a server (shares its resources with others). This way, P2P systems offer scalability and fault tolerance by design.

Instead of forming overlay networks atop the current Internet, the ICN paradigm proposes for a radical shift from the current host-centric communication model into a content-centric one. More precisely, ICN suggests to redesign the Internet from scratch as a content-centric network. This concept was the basis for several ICN architectures proposed in the last years. Notable examples include: Data Oriented Network Architecture (DONA) [KCC⁺07], Named-Data Networking (NDN) [JST⁺09], Publish/Subscribe Internet Routing Paradigm (PSIRP) [LVT10], and Network of Information (NetInf) [Dan09]. Despite their differences, those architectures are based on two common ideas: (i) networking named contents and (ii) in-network caching.

1.1 Problem Statements

We deal in this dissertation with two systems from the field of content distribution: (i) Kademlia [MM02], the most actively used P2P system, and (ii) NDN [JST⁺09], the most extensively researched ICN proposal and widely considered a potential architecture for the future Internet. In spite of the success and popularity of Kademlia-type systems and the great promises of NDN, several research problems still need to be addressed in both systems. We address in this dissertation three of those problems: (i) characterizing topological and routing table properties of Kademlia, (ii) coordinating cache management in NDN, and (iii) defending against DDoS attacks in NDN. We state and motivate the three problems in this section.

1.1.1 Characterizing Topological and Routing Table Properties of Kademlia

Topological and routing table properties represent predominant concerns when designing P2P systems with outstanding performance, reliability, and availability features. Accurate characterization of these properties is requisite for evaluating the performance and robustness of these systems, for proposing accurate models, and for suggesting design improvements.

In particular, the resilience of graph connectivity, both to random failures and to targeted attacks, is a direct measure for the system's availability and reliability for communications [DNF⁺08, DGLL07]. Both completeness of routing tables and diversity of node's contacts over the respective identifier space sections influence the routing performance [SR06a]. The degree distribution describes the structure of the topology and it also relates to the resilience and routing table properties.

We focus on Kademlia, the most popular P2P system today. The aforementioned properties were analysed in Kademlia-type systems so far based on partial measurements or oversimplified simulations. These simulations implemented small system sizes and ignored user misbehaviour and dynamics. Instead, accurate characterization can be derived from analysis of graph snapshots of real Kademlia-type systems. However, such snapshots were not provided by previous measurements, and capturing them is challenging since it involves collection of large amount of information almost instantly from distributed, large [WK13, SS13a], and rapidly changing [SENB09] systems.

The first problem which we address in this dissertation is threefold: (i) capturing accurate graph snapshots of an active Kademlia-type system, (ii) characterizing the aforementioned properties using the captured snapshots, and (iii) utilizing the obtained knowledge for improving the system if possible.

1.1.2 Coordinating Cache Management in NDN

NDN implements in-network caching to improve content delivery as well as to diminish the pressure on the network bandwidth. NDN applies an autonomous cache management scheme. That is, each node takes independent caching-related decisions according to its own view of the requested and cached contents. Such a scheme, although simple, is not efficient [RR11,SLYJ13]. In particular, it results in unnecessary large cache redundancy as well as in suboptimal selection for contents which are given priority for caching. In addition, routing of content requests in NDN is cache-ignorant. Hence, a cached copy of the requested content can be used only if it is located on the default route of the corresponding request packet.

Consequently, cache coordination has been the subject of several recent studies (*e.g.* [SLYJ13,BKST13,SFT13]). The solutions, however, are either of limited benefit since their designs do not enable to efficiently leverage in-network caches or impractical due to their high coordination overhead.

The second problem which we address in this dissertation is to develop a coordinated cache management scheme for NDN that is efficient and practical.

1.1.3 Defending Against DDoS Attacks in NDN

NDN is vulnerable to new types of Distributed Denial-of-Service (DDoS) attacks [AHZ15]. Among the identified attacks, the Interest Flooding Attack (IFA) can significantly degrade the QoE of users [AMM⁺13,DWFL13]. The adversary in IFA exploits two properties of NDN: (i) routing based on longest name-prefix match and (ii) storing a forwarding state per content request (known as interest packet) in the so-called Pending Interest Table (PIT). More precisely, the adversary issues interest packets for non-existent contents. Consequently, one PIT entry is created per interest packet in each node on the path and stays there till it eventually expires. Overflowing PITs leads to dropping part of legitimate interest packets. Despite the considerable number of proposed defence mechanisms against IFA (see [AMM⁺13,DWFL13] and the references therein), they are not highly effective, and part of them incurs high overhead.

IFAs can be mitigated effectively when they are detected and mitigated quickly close to potential attack sources based on timely and aggregated attack-related information. The third problem which we address is to realize such a solution with low overhead.

1.2 Research Methodology

We apply a unified research methodology to address the three aforementioned problems. The methodology, as shown in Figure 1.2, consists of three steps:

Firstly, we develop measurement approaches that are able to address the challenge of collecting large volume of system-wide information quickly from systems that are large and dynamic. Secondly, we use the developed measurement approaches to measure relevant properties. Thirdly, we use the knowledge that is gained from the measurements either to suggest representative system models and design improvements or to adapt the system operation.

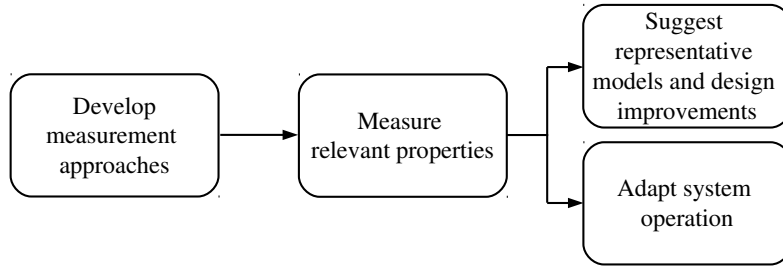


Figure 1.2: Research methodology

1.3 Contributions

We address in this dissertation the three aforementioned problems through five scientific contributions. The contributions were published in the proceedings of peer-reviewed conferences and workshops.

We structure the dissertation in two parts: in Part I, we address the problem of characterizing topological and routing table properties of Kademlia-type systems. After that, in Part II, we address both the problem of coordinating cache management and the problem of defending against IFA in NDN.

Part I consists of the following three contributions:

1. We develop KadSpider, a crawler for capturing accurate graph snapshots of KAD (a popular Kademlia-type system).
2. We present and discuss the results of an extensive characterization study for Kademlia-type systems using KAD as a case study. The analysis is based on both real graph snapshots captured by KadSpider and synthetic graphs generated by simulations.
3. Using real KAD routing tables collected by KadSpider, we analyse a previously disregarded property: the diversity of contacts within each routing table section over the respective sections of the identifier space. We also show that maximizing this diversity improves the standard routing performance. We consequently propose a modified neighbour selection scheme that maximizes the aforementioned diversity. The new scheme causes negligible

overhead, and it is compatible both with the standard protocol as well as with former protocol improvements. We evaluate the new scheme via derivations of our theoretical model for Kademlia routing [RSS15], extensive simulations of three Kademlia-type systems, and measurements on KAD clients.

The three contributions above were presented and published in [SS13a], [SRS14a]¹, and [SRS14b], respectively.

Part II is constituted of the following two contributions:

4. We present the design and evaluation of CoMon, a framework for network-wide coordination in NDN. CoMon assigns monitoring and other coordination-related tasks to few nodes only. We develop a heuristic to select these nodes such that they capture the entire network traffic, at an early stage. As part of this contribution, we employ CoMon to coordinate caching-related decisions and to realize cache-aware routing on Autonomous System (AS)-wide scale. We evaluate the caching efficiency which can be achieved with CoMon as well as its signalling and hop count overhead through simulations applying realistic settings, comparing CoMon both to the original system and to notable prior solutions.
5. We develop a new defence mechanism against IFA. The mechanism detects and mitigates IFAs in a coordinated way based on aggregated attack-related information. In practice, our solution adapts the design concepts of CoMon to achieve effective and low-overhead coordinated defence. We evaluate the solution via an extensive simulation study.

The idea and preliminary evaluation of CoMon first were presented and published as a poster paper [SSS14]. Then, the detailed design and evaluation of CoMon (as a framework for coordinating caching-related decisions) were published in [SS15]. Similarly, the last contribution was first introduced in [SWS15b] and then detailed in [SWS15a].

1.4 Organization

The remainder of this dissertation is structured as follows. Part I, presenting the characterization study of Kademlia-type systems, is organized in four chapters: in Chapter 2, we give an overview of the key design elements and operation primitives of Kademlia and three of its notable derivatives, with which we deal in this part. Then, in Chapter 3, we present the design and evaluation of KadSpider, our

¹ Best student paper in IEEE ISCC 2014.

measurement approach for capturing graph snapshots of KAD. Next, we present and discuss the characterization results of measured and synthetic KAD graphs in Chapter 4. After that, we describe and evaluate our new neighbour selection scheme for Kademia-type systems in Chapter 5.

We document in Part II our work on improving the NDN architecture in three chapters: we start by giving an overview of NDN, highlighting the design details which directly relate to our work in Chapter 6. Next, in Chapter 7, we present and evaluate CoMon, our solution for coordinating caching-related decisions in NDN. After that, in Chapter 8, we describe how we adapt and employ CoMon for defending against IFA. Finally, we summarize the dissertation and suggest directions for future work in Chapter 9.

1.5 Use of Published Work

We adapt portions of this dissertation from the material of the aforementioned co-authored publications with permission from the corresponding co-authors. In particular, Part I includes material adapted from [SS13a, SRS14a, SRS14b, RSS15], while [SSS14, SS15, SWS15b, SWS15a] constitute the three chapters of Part II. We add proper citations whenever the adapted material is not originally written or not plotted by the author of this dissertation.

Part I

Understanding and Improving Kademlia

Overview of Kademlia-type Systems

Peer-to-Peer (P2P) systems can be categorized as either unstructured or structured. On the one hand, in unstructured P2P systems connections are established arbitrarily, and nodes rely on query flooding to search for objects (*e.g.* files). Each queried node answers with a list of matching objects. Since flooding is extremely inefficient, unstructured P2P systems usually enforce a hop count limit on flooded queries. Such a limit, however, decreases the chances to find requested objects.

In structured P2P systems, on the other hand, routing is deterministic, and nodes adhere to a certain structure that defines the relationship between nodes and object locations. This design guarantees that requested objects, if they exist, are found. The popular form of structured P2P systems is the so-called Distributed Hash Table (DHT). Similar to the hash table, the DHT maps objects in the form of key-value pairs to nodes. To resolve this mapping, the DHT facilitates a decentralized message routing to the respective node in a very robust fashion.

In this part of the dissertation, we exclusively study Kademlia [MM02], the most popular and most widely deployed DHT, today. We give in this chapter an overview of the concepts that Kademlia-type systems are based on.

2.1 The Original Kademlia

Kademlia is implemented as an overlay network for discovering peers and objects in highly popular applications such as BitTorrent [bt] and eMule [emu]. It also was experimented as a communication overlay for video streaming applications [Jim13] and botnets [HSD⁺08, SKK08].

Kademlia uses a b -bit ($b = 160$) identifier space from which the identifiers of nodes and objects are assigned. It implements key-based routing and storage of key-value (identifier-object) pairs. The nodes at the closest distance to a content's identifier are responsible for storing it. The distance between two identifiers is defined as the XOR of their values.

Each node v stores the identifiers and the addresses of other nodes (also called neighbours or contacts) in a b -level tree-like routing table. Each level in the routing table consists of the so-called k -buckets, where each bucket stores up to k known contacts sharing a common prefix with v 's identifier. More precisely, for each routing table level i the stored contacts are of distance between 2^i and 2^{i+1} from v 's identifier. Kademlia uses $k > 1$ to improve the robustness of routing information in the presence of churn (*i.e.* independent arrival and departure of nodes). Contacts within each bucket are kept sorted from least-recently seen to most-recently seen. Information stored in the routing table becomes outdated, or stale, when the respective nodes leave the system.

Kademlia applies a least-recently seen eviction policy in the routing table buckets. The only exception is that live nodes are not evicted from the routing table. This exception is driven by the authors' analysis of the Gnutella measurement trace data collected by Saroiu et al. [SGG01]. In particular, the authors in [MM02] calculated the percentage of nodes that stay online one more hour as a function of the current uptime. They found that the longer a node has been online, the more likely it is to stay online another hour. Accordingly, keeping the oldest live contacts increases the probability that they will remain online.

Kademlia implements a key-based greedy routing protocol (also called the lookup protocol): to route a message from a node v to a target x , v picks the α closest contacts to x and sends them lookup requests in parallel. Each queried contact, if online, replies with a set of β contacts that are locally known as being closest to x , thus extending v 's set of candidate contacts. This process iterates either until no further contacts closer to x are discovered or until a timeout is held. In this protocol, the lookup's hop count refers to the number of edges on the shortest path traversed during a lookup, where each hop represents a transition from a set of queried contacts either to another set of queried contacts or to lookup termination [RSS15]. The original Kademlia paper suggests to use $k = 20$ and $\alpha = 3$.

With the above described designs of the routing table and the routing protocol, Kademlia realizes highly efficient routing. In particular, the number of nodes that are contacted during a lookup as well as the number of contacts every node needs to store grow only logarithmically in the system size [MM02].

The above design is the basis for a family of Kademlia-type systems. We focus in this dissertation on three variations of Kademlia: the mainline DHT (MDHT), a modified version of mainline called iMDHT, and KAD. We deal with MDHT and KAD motivated by their high popularity: they are used by about 27 million [WK13] and half a million [SS13a] simultaneous users, respectively. iMDHT was proposed in [JOK11] to improve MDHT. The authors showed that iMDHT improves the routing performance of MDHT remarkably. Next, we describe the specifications of the three systems.

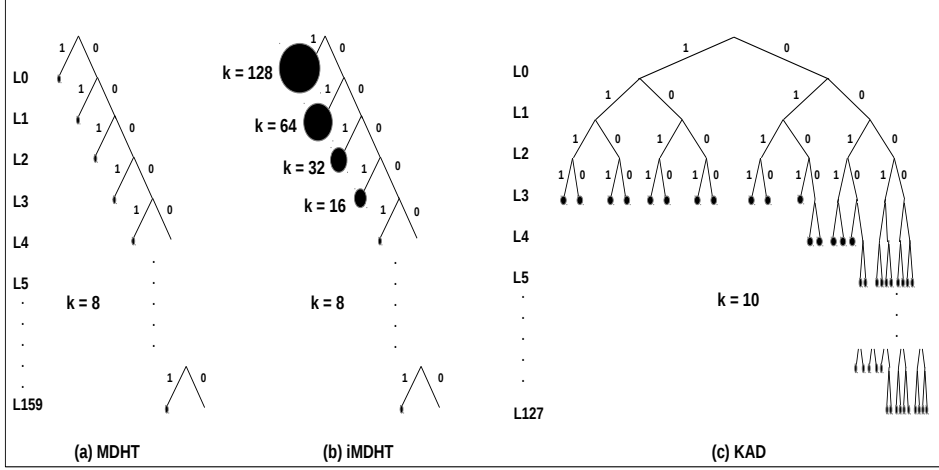


Figure 2.1: Routing table structures of three Kademlia-type systems (adapted from [WTCT⁺08]): (a) MDHT, (b) iMDHT, and (c) KAD. Each routing table encodes the distances in the identifier space: the top part stores the farthest contacts, while the bottom part stores the closest ones. Each leaf represents a bucket of contacts.

2.2 MDHT and iMDHT

The mainline implementation of BitTorrent (MDHT) integrates a Kademlia-type overlay for node discovery. In MDHT, the routing table (Figure 2.1a) includes a single k -bucket per level. uTorrent [ut], the most popular MDHT implementation, uses $k = 8$, $\alpha = 4$, and $\beta = 1$ [JOK11].

Considering the fraction of the identifier space that is covered by each routing table level i , Jimenez et al. [JOK11] introduced a variation of MDHT implementing variable bucket sizes (iMDHT) to increase the distance reduction at each hop. The bucket sizes are chosen to be 128, 64, 32, and 16 for the buckets at levels 0, 1, 2, and 3, respectively, and 8 for the rest (Figure 2.1b). Both MDHT and iMDHT use $b = 160$.

2.3 KAD

KAD¹ is used by the popular file-sharing application eMule. It uses $b = 128$, $k = 10$, $\alpha = 3$, and $\beta \in \{2, 4, 11\}$ depending on the search type. KAD implements a different routing table structure: as shown in Figure 2.1c, starting from the fourth level, the routing table includes multiple buckets per level, grouping contacts according to the first $l \in \{3, 4\}$ bits after the first varying bit. Consequently, the bit gain (*i.e.* the difference between the common prefix length of the current hop and the next hop to x) is at least l .

¹ We describe here only the specifications of KAD that are directly related to our work. For a detailed description of KAD, the reader is referred to [Bru06] and [Mys06].

The routing process in KAD relies on two UDP-based messages²: `Kademlia_request` (`KAD_REQ`) and `Kademlia_response` (`KAD_RES`). Among other fields, the `KAD_REQ` message specifies β , the target identifier field, as well as the contact information of the destination node represented by its KAD identifier (`KID`), IP address, and UDP port.

2.3.1 Routing Table Maintenance

In order to mitigate the effect of churn, each node performs maintenance for its routing table by running two processes. The first process aims at populating buckets that have many empty slots. More precisely, each bucket is checked every hour if it contains less than three contacts, or if it can be divided into two buckets. In either case, the process looks for an identifier selected uniformly at random (but within the corresponding identifier range) if the bucket has a matching prefix length of less than five bits with the node's identifier. Otherwise, the process looks for the node's identifier itself. The returned contacts are then inserted in the corresponding buckets (or in the resulted children buckets).

The second process aims to keep the routing information up-to-date. For this, it checks whether the stored contacts are still responsive or not, and removes the unresponsive (*i.e.* stale) ones. More precisely, the process associates each contact with an expiry time, and checks it periodically according to its life time, where long-lived contacts are checked less frequently than newer ones. This design is based on the aforementioned observation which states that the longer a contact has been online the more likely it remains online in the future. The node sends to each expired contact a `Hello_request` message, and removes it from the routing table if it does not respond within two minutes. Contact removals may result in merging buckets.

2.3.2 Bootstrapping

In case a node does not have any active contact in its routing table, either because the node is new or because it did not connect to KAD for a long time, the bootstrap process should be executed. To do so, the node first has to know one or more active contacts³. The node then sends to each of the learned contacts a KAD's `Bootstrap_request`. Each queried contact, if online, answers with a `Bootstrap_response` message containing 20 contacts selected uniformly at random from its own routing table.

² The nodes that cannot directly receive unsolicited UDP packets (*e.g.* because they are located behind Network Address Translators (NATs)) are not part of the DHT [SR06b,emu]. Such nodes still can participate in the file-sharing activities but only through directly connected nodes called buddies.

³ These contacts are usually obtained from the client's website [emu].

2.3.3 Populating the Routing Table

Contacts in KAD are added to a node's routing table in the following occasions. When the node starts-up, it reads contacts from a file storing the contacts learned during the last time the node was online. Contacts also are added during bootstrapping as well as during routing table maintenance. Another case is to add the contacts from which the node receives `Bootstrap_requests` or `Hello_requests`, in case the corresponding buckets have free slots.

2.3.4 Protection Against Flooding Attacks

The recent version of KAD (called KAD2) implements the so-called flooding protection mechanism. This mechanism aims both at protecting the nodes against flooding of request messages as well as at detecting and dropping unsolicited response messages.

With this mechanism, KAD sets a threshold on the number of messages that can be received from a node within a specific period of time. For instance, a node is allowed to send another node one `KAD_REQ` within six seconds. Messages exceeding this limit are considered malicious and dropped. Cases of repeated misbehaviour are treated by banning the sender permanently.

2.4 Summary

We reviewed in this chapter the key design elements and operation primitives of Kademlia, the most widely deployed P2P system, with three notable examples of its real implementations. Kademlia maps key-value pairs to nodes. It resolves the mapping through a decentralized message routing to the corresponding node in a robust way. Such a resolution is highly efficient: both the number of nodes that are contacted to resolve a discovery request as well as the state maintained by every node grow only logarithmically in the system population. This high efficiency is mainly attributed to the tree-like routing table as well as to the greedy and parallel routing protocol.

With the goal to better understand and improve Kademlia-type systems, we present in the next three chapters an extensive characterization study on these systems. In particular, we focus on topological and routing table properties relating to routing performance and the robustness of the network graph for communications.

Capturing Accurate Snapshots of the KAD Graph

Kademlia and its derivatives are the most popular and the most widely deployed P2P systems, today. We aim in this part of the dissertation to contribute to the understanding of these systems' performance and robustness by studying relevant topological and routing table properties. In particular, we focus on five properties: (i) the diversity of routing information over the corresponding identifier space sections, (ii) the completeness of routing information with respect to global knowledge of the system graph, (iii) the degree distribution, (iv) the graph resilience in face of random node departures, and (v) the graph resilience in face of targeted node removals. Accurate characterization of these properties is substantial to evaluating and improving these systems and the protocols that work over them, as well as to deriving representative models.

To the best of our knowledge, we are the first to analyse the properties (i) and (ii). As for the properties (iii)–(v), they were studied based on simulations only (*e.g.* [DNF⁺08, HYEN09]). The simulative studies, however, applied small system sizes and oversimplified churn settings, and they disregarded user misbehaviour. Instead, with the goal to achieve more accurate and realistic results, we propose to study these properties in a real system using snapshots of the system graph. Such snapshots were not provided earlier. In addition, capturing them is highly challenging [SRS08], because it requires to download large amount of highly dynamic and distributed information [SR06b, SENB07] almost instantly.

We address this challenge by KadSpider, our own crawler for capturing representative graph snapshots of KAD. A crawler is a widely used term in the P2P and network measurements literature. It refers to a measurement tool that walks a P2P system querying every node for its neighbours (or part of them) to iteratively explore the entire graph [SR09]. We chose KAD for our analysis because it is a popular implementation of Kademlia [SENB09, WC11].

We describe in this chapter the design and evaluation of KadSpider, and we also give an overview of our dataset.

3.1 Related Work

There was a considerable number of measurement studies for several P2P systems, in recent years. We restrict the discussion in this section to studies that developed crawlers for Kademlia-type systems. We group prior Kademlia crawlers by the type of information they collect in two groups.

The first group includes crawlers that download partial routing tables either from all discovered nodes or part of them. Notable examples include the following: Kutzner et al. [KF05] introduced a crawler for Overnet (a Kademlia-type system) to measure the network size, availability of nodes and their distribution in the underlay network, as well as the distribution of nodes' round-trip-times (RTTs). BitMon [JADH10] was designed to measure part of the Mainline DHT to estimate several network usage statistics. Closer to our work, some studies crawled KAD. For instance, Cruiser [SR05a] measured churn behaviour in KAD [SR06b]. Blizzard [SENB09] measured several properties in KAD such as lengths of user sessions, geographical distribution of users, and identity aliasing. Wu and Chen [WC11] developed a crawler which improved on Blizzard. In particular, they deployed the crawler in a distributed way and slightly improved the crawling algorithm. A similar crawler is Rainbow [LMCC10] which was developed to measure information about the nodes, client's software, and popularity of shared files. Yu et al. [YLX⁺11] developed a crawler called Rememj to analyse lookup traffic.

The second group of crawlers includes those that download entire routing tables, but from a fraction of DHT nodes only. Such crawlers were developed, for instance, by Wang et al. [WTCT⁺08] to perform certain attacks on KAD, by Kang et al. [KCTHK09] to study the replication performance, and by Stutzbach and Rejaie [SR06a] to study the completeness and freshness of routing information.

From the design's point of view, the idea of the aforementioned crawlers is the same: the crawler progressively explores the system by querying known contacts for their neighbours, adds discovered contacts to the list of known nodes, and iterates over the list till nearly no new contacts are discovered.

Despite the important knowledge provided by the aforementioned studies, none of them captured snapshots of the system graph. Such snapshots, however, are necessary to analyse the completeness of routing tables (with respect to global system knowledge), the in-degree and degree distributions, as well as the two aforementioned types of graph resilience.

Capturing accurate graph snapshots is more challenging than previous measurements: it requires discovering all online nodes and download their entire

routing tables nearly instantly and without information losses. Previous crawlers are not designed to conduct such measurements in an efficient way. Hence, a new crawler, that improves on previous designs and deployments, is necessary to capture accurate graph snapshots (for our characterization study).

3.2 Crawling Task and Challenges

Definition 1. A graph $G = (V, E)$ consists of a set V of nodes (or vertices) and a set $E \subset V \times V$ of edges (or links).

The KAD graph is unidirectional, meaning that $(u, v) \in E$ (an entry in u 's routing table towards v) does not imply $(v, u) \in E$. Consequently, the task of capturing a snapshot of KAD graph implies two interleaving steps: (i) discovering online nodes and (ii) downloading their routing tables.

Capturing exact (*i.e.* instant and complete) snapshots of the KAD graph is technically infeasible due to the large size and high dynamics of KAD [SR06b, SENB09], in addition to the large size of its routing table (Figure 2.1c). That is to say, the aforementioned crawling task will involve sending, receiving, and processing massive amount of network traffic within a short period of time. Consequently, very high crawling rates will cause excessive network congestions. Given that KAD crawlers mainly rely on the standard UDP-based¹ routing packets, it is to be expected that part of these packets will be lost. Losses of response packets, in turn, affects the completeness of captured snapshots. Furthermore, it is important that the crawling rate should not violate KAD's flooding protection mechanism (Section 2.3.4). Otherwise, the target nodes will drop (at least part of) the crawling queries.

Instead, we guide the design of KadSpider by two more practical requirements:

- R1) Graph changes during crawling (due to churn) should be kept low.
- R2) Graph information should be nearly complete.

3.3 Design and Implementation

We describe in this section KadSpider's design and implementation details.

3.3.1 Advanced Crawling Techniques

We describe here four design and deployment techniques implemented in KadSpider: (i) dividing the routing table tree into sub-trees of buckets, (ii) avoiding querying

¹ UDP (User Datagram Protocol) uses a simple connectionless transmission that does not give guarantee of delivery.

empty buckets, (iii) adaptive crawling rate, and (iv) distributing the crawling task. The first two techniques exploit KAD design better than prior crawlers, aiming to outperform them in terms of the time and signalling overhead of downloading routing tables. Note that reducing the signalling overhead leads to lower network congestions, thus to lower losses of crawling messages (*i.e.* higher completeness). The third and fourth techniques aim to improve the completeness of the captured snapshots.

Dividing the Routing Table Tree into Sub-trees of Buckets

KadSpider relies on the standard routing packets of KAD: KAD_REQ and KAD_RES. Each routing query (hereafter, crawling query or query) is used to download a certain region of the routing table. Prior studies (*e.g.* [SCB10]) assumed that the maximum number of contacts that can be returned in a KAD_RES (*i.e.* β) is 11. Consequently, their crawlers were designed to target one bucket (containing up to 10 contacts) only per query.

However, β in the client code can take any value between 0 – 31. This means that more contacts can be returned per query, thus the total number of queries for downloading an entire routing table can be reduced. However, this requires to prepare the queries such that the overlaps of returned contacts is minimized.

KadSpider realizes this as follows. It divides the routing table tree into 256 non-overlapping sub-trees: 129 two-bucket sub-trees and 127 three-bucket sub-trees. In Figure 3.1, the sub-trees 0 and 8 are examples of two-bucket and three-bucket sub-trees, respectively. The sub-trees are numbered from 0 to 255, starting from the routing table’s top-left part towards its bottom-right edge (which contains the node’s closest contacts). Consequently, KadSpider uses a single crawling query to download a sub-tree of buckets, rather than a single bucket only per query. Given that a sub-tree of buckets contains on average about 2.5 buckets, KadSpider (only by using this technique) accelerates the time for downloading routing tables and reduces the number of crawling messages, both by about 2.5 times.

In compliance with this design, KadSpider prepares KAD_REQs as follows: with a maximal bucket size of 10 in KAD, KadSpider sets the β field (number of requested contacts) to 30 for targeting three-bucket sub-trees or to 20 for targeting two-bucket sub-trees. As for the target-ID field, its value is computed as: target node’s KID \oplus sub-tree root’s ID, where the sub-tree root’s ID is a 128-bit value representing the position of the sub-tree’s root in the routing table.

Avoiding Querying Empty Buckets

The measurements of Stutzbach and Rejaie on KAD routing tables [SR06a] showed that a large part of routing table buckets are empty, particularly at low levels (due to the limited number of matching contacts in the system). This also likely

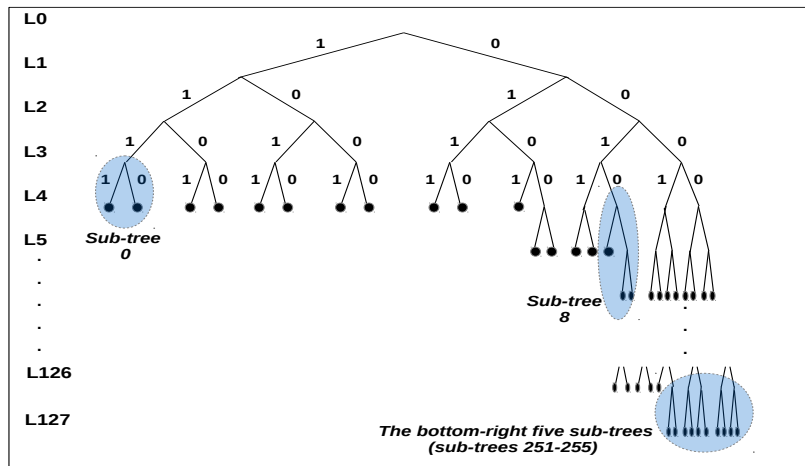


Figure 3.1: Exemplary sub-trees in a KAD routing table. The top-left part stores the farthest contacts, while the bottom-right part stores the closest ones. "0" and "8" are examples of two-bucket and three-bucket sub-trees, respectively. (The original routing table figure is adapted from [WTCT⁺08].)

means that there are sub-trees of empty buckets. According to this observation, we implement a technique in KadSpider that attempts to avoid querying sub-trees of empty buckets.

The technique works as follows: the crawler first sends five KAD_REQs to discover one contact in each of the five lowest routing table sub-trees (251 – 255), *i.e.* the ones that store the closest contacts (Figure 3.1). Next, the crawler shifts to the top of the tree, and starts downloading sub-trees of buckets from top-left towards bottom-right (0 – 255), in order, till it encounters the previously reported five closest contacts again. At that moment, KadSpider stops sending KAD_REQs to the target node because there are no further contacts, deeper in the tree, to discover. The more sub-trees that are skipped with this technique, the faster the crawling task and the lower its signalling overhead.

Adaptive Crawling Rate

With this technique, KadSpider tunes the crawling rate (*i.e.* the rate at which crawling queries are sent) according to the observed packet losses. In practice, KadSpider continuously measures packet losses by parsing network system files² that report relevant traffic-related values, and tunes the crawling rate according to the ratio of lost packets.

² "/proc/net/dev" and "/proc/net/udp" in Linux operating systems.

Distributing the Crawling Task

In principle, there are two main approaches to crawl a system: centralized or distributed. In the centralized approach (*e.g.* Blizzard [SENB09]), a single crawling machine performs the entire crawling task. This approach is appropriate when the crawling overhead (in terms of processing, memory, and bandwidth) can be handled by a single machine. As discussed above, our crawling task involves sending and receiving large amount of data and on-the-fly computations within a short period of time. Conducting our measurements on a single machine, we faced several problems such as software crashes and large amount of packet losses due to network congestions. Therefore, we chose to distribute our crawling task over multiple machines locating at different networks, as illustrated in Figure 3.2.

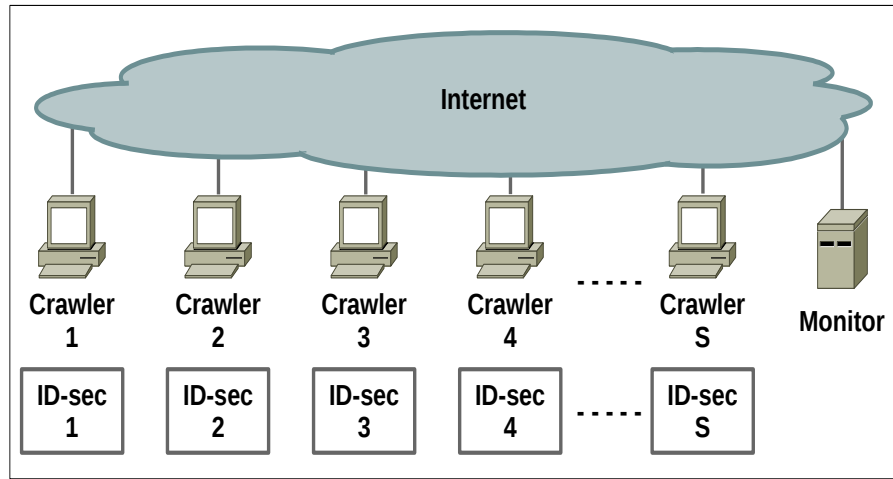


Figure 3.2: Crawling architecture of KadSpider: the crawling task is distributed over equal crawling machines, each of them crawls a certain (non-overlapping) section of the identifier space.

In more detail, since KAD is assumed to distribute nodes' KIDs uniformly over the identifier space [SENB09], we partition the identifier space of KAD into equal non-overlapping sections, and distribute them evenly on the available crawling machines, assuming they have the same resources. Each crawling machine works independently and exclusively on its own identifier space section. The machines thus do not require to synchronize among each other. This way, the crawling overhead is distributed over the machines, which facilitates higher crawling rates and simultaneously reduces network congestions thus lowers the losses of crawling messages.

In addition, we used a monitoring machine to check whether the primary crawling machines work properly or not. If not, malfunctioning crawling machines are automatically substituted by other (backup) machines.

3.3.2 Algorithmic Design

We describe here the algorithmic design of KadSpider which consists of three asynchronous threads (Algorithms 1 – 3). The main data structures used by the threads are the following:

- *k-Node*: struct{KID, IP address, UDP port}
- *close*: struct{Node (k-Node), found_again (bool)}
- *a-Node*: struct{Node (k-Node), Last_packet1, Last_packet2, Last_time, closest[] (close), Stop_send (bool)}
 - ▷ *Last_pkt1* and *Last_pkt2* specifies the index of the last KAD_REQ sent by *Send-1* and *Send-2*, respectively, to the corresponding node (*i.e.* the last queried routing table sub-tree). *Last_time* specifies the time when the last query was sent to the corresponding node, and *closest* specifies the node's closest five contacts.
- *known-Nodes (KN)*
 - ▷ A file that stores contact information of 1000 nodes (a list of *k-Node* elements) from the previous crawl. The contacts are selected such that their KIDs are uniformly distributed over the identifier space.
- *current-Nodes (CN)*
 - ▷ List of *a-Node* elements storing information of nodes discovered during the crawl. When a new crawl starts, *CN* is initialized with the nodes information stored in the *KN* file. The *Last_packet1*, *Last_packet2*, *Last_time*, and *Stop_send* fields are initialized with: 251, 0, the current time, and *false*, respectively. This initialization applies for all *CN*'s newly added entries.

The *Send-1* thread (Algorithm 1) is responsible for downloading the closest five contacts of each responding node in *CN*. It does so by iterating over the *CN* list and sending each node five KAD_REQs, using $\beta = 1$, targeting the routing table sub-trees 251 – 255.

The *Send-2* thread (Algorithm 2) is responsible for downloading routing tables of all responding nodes in *CN*. It iterates over the *CN* list, and for each node, the thread sends KAD_REQs targeting the routing table sub-trees 0 – 255 in order. The thread skips the nodes whose *Stop_send* field is set to *true*; this means either that all 255 sub-trees are already queried or that the five closest contacts are returned twice (in two different KAD_RESs: queried by *Send-1* and *Send-2*, respectively). In order to not violate KAD's flooding protection mechanism, Algorithm 1 (line: 3) and Algorithm 2 (line: 4) restrict the query rate for each target node to one query every six seconds.

Algorithm 1 Send-1

```

1: while true do
2:   for each  $m \in \text{CN}$  do
3:     if  $\text{CN}[m].\text{Last\_packet1} \leq 255 \wedge \text{diff}(\text{CN}[m].\text{Last\_time}, \text{now}) \geq 6$  then
4:       Send_KAD_REQ( $\text{CN}[m].\text{Last\_packet1}$ )
5:       ++  $\text{CN}[m].\text{Last\_packet1}$ 
6:        $\text{CN}[m].\text{Last\_time} \leftarrow \text{now}$ 
7:     end if
8:   end for
9: end while

```

Algorithm 2 Send-2

```

1: while true do
2:   for each  $m \in \text{CN}$  do
3:     if Not  $\text{CN}[m].\text{Stop\_send}$  then
4:       if  $\text{diff}(\text{CN}[m].\text{Last\_time}, \text{now}) \geq 6$  then
5:         Send_KAD_REQ( $\text{CN}[m].\text{Last\_packet2}$ )
6:         ++  $\text{CN}[m].\text{Last\_packet2}$ 
7:         if  $\text{CN}[m].\text{Last\_packet2} == 256$  then
8:            $\text{CN}[m].\text{Stop\_send} \leftarrow \text{true}$ 
9:         end if
10:         $\text{CN}[m].\text{Last\_time} \leftarrow \text{now}$ 
11:       end if
12:     end if
13:   end for
14: end while

```

The *Receive* thread (Algorithm 3) is responsible for handling incoming KAD_RESs, which includes the following steps: First, it extracts new contacts from the received KAD_RESs, and inserts them to the *CN* list. Second, it extracts the information of nodes' closest contacts, and accordingly updates the *closest* array as well as the *Stop_send* field (in case any of the respective conditions is met). Third, it extracts graph information that is returned in the KAD_RES, and records it in a log file (hereafter, *graph_file*). More precisely, each KAD_RES contains information of β contacts, each of them represents a graph edge originating from the sender node towards another node in the graph.

Each edge in the graph is represented in the *graph_file* by a five-field record, specifying the source node's IP address and UDP port, and the destination node's IP address, UDP port, and KID. At the end of the crawl, the *graph_file* is used to reconstruct the corresponding graph snapshot.

Due to churn in KAD, the crawl needs to be stopped as soon as the rate of discovering new contacts becomes very low. Otherwise, the accuracy of the captured snapshot becomes low [SR05b]. In KadSpider, we use the crawl

Algorithm 3 Receive

```

1: while true do
2:   for each received KAD_REQ RES do
3:     for each node M  $\in$  RES do
4:       graph_file.add(Edge: Sender  $\rightarrow$  M)
5:       if M  $\notin$  CN then
6:         CN.add(M)
7:       end if
8:       if Not CN[c].Stop_send then  $\triangleright$  c: corresponding node
9:         if  $\beta == 1$  then  $\triangleright$  Response to a query from Send-1
10:          CN[c].closest.add(M, false)
11:        else  $\triangleright$  Response to a query from Send-2
12:          if M  $\in$  CN[c].closest then
13:            CN[c].closest[x].found_again  $\leftarrow$  true
14:             $\triangleright$  x: matching closest contact entry
15:            if CN[c].closest.found_again  $\forall$  five closest contacts then
16:              CN[c].Stop_send  $\leftarrow$  true
17:            end if
18:          end if
19:        end if
20:      end if
21:    end for
22:  end for
23: end while

```

termination technique of Blizzard [SENB07]. More precisely, KadSpider stops querying nodes when 99% of the active nodes in the nodes list are queried completely. After that, the software waits 60 seconds for late responses before terminating the crawl completely. We do not include these details in the listed algorithms.

3.4 Dataset and Performance Results

In this section, we first describe our measurements setup and dataset of captured KAD graphs (Section 3.4.1). After that, we evaluate the performance of KadSpider with respect to the predefined design requirements (Section 3.2).

3.4.1 Measurement Setup and Dataset Description

We used the German-Lab (G-Lab) testbed [gla] for the final deployment of KadSpider. In particular, we used 64 primary crawling machines, 20 backup crawling machines, and one monitoring machine. Accordingly, we partitioned the

identifier space of KAD evenly into 64 non-overlapping sections.

Our dataset contains hundreds of partial snapshots and 65 snapshots captured in the full crawls (*i.e.* those that targeted the entire KAD graph). We captured these snapshots using KadSpider at different times over the period between April 2012 and September 2013.

Table 3.1 gives an overview of the snapshots captured in the full crawls³. In particular, the table shows the maximum, minimum, median, average, and standard deviation values of (i) concurrent online KAD nodes, (ii) all discovered KAD nodes (including both online and offline nodes), and (iii) number of crawling queries that are needed to download an entire routing table. We count the nodes by their unique UDP socket identities (*i.e.* $\langle \text{IP address, UDP Port} \rangle$). Our statistics are based on the information contained in the KAD_RESs received from the nodes that remained online since they were discovered till the end of the crawl⁴. Since NATed nodes do not participate in the DHT (see Section 2.3), we do not count them in these statistics, nor does their absence affect the accuracy of our snapshots.

Table 3.1: Overview of 65 KAD graph snapshots captured in the full crawls over the period between April 2012 and September 2013

	Conc. online nodes	All discovered nodes	Sent KAD_REQs
Minimum	352,390	3,521,470	23
Maximum	508,059	4,705,002	74
Median	470,981	3,826,106	33
Average	452,740.35	3,812,030.43	31.92
Std. dev.	48,709.12	212,641.71	7.59

The KAD population reported in the full crawls, although smaller than prior results (*e.g.* Rememj [YLX⁺11] reported 1.2 – 1.8 million nodes in 2009), indicates that KAD is still highly active and popular, thus was a proper choice for our characterization study.

We plot in Figure 3.3 the geographical distribution of KAD nodes showing the top ten countries (in terms of KAD usage), both for the concurrent online nodes as well as for all discovered nodes. The results represent an exemplary full crawl conducted on 12.09.2013 (started at 20:00 GMT). This distribution is similar to what we observed in the other snapshots, and it also resembles the distributions

³ In cooperation with the developer of Blizzard [SENB09], we validated our measured KAD population results via crawling tests performed by Blizzard. Very close to our results, Blizzard estimated the population of concurrent online KAD nodes in August 2012 by about 500,000.

⁴ This applies for all results. This information also is used for reconstructing the graph snapshots from the *graph_file*.

reported by previous studies [SENB09]. It can be seen that the largest fraction of users are from China, followed by users from west European countries.

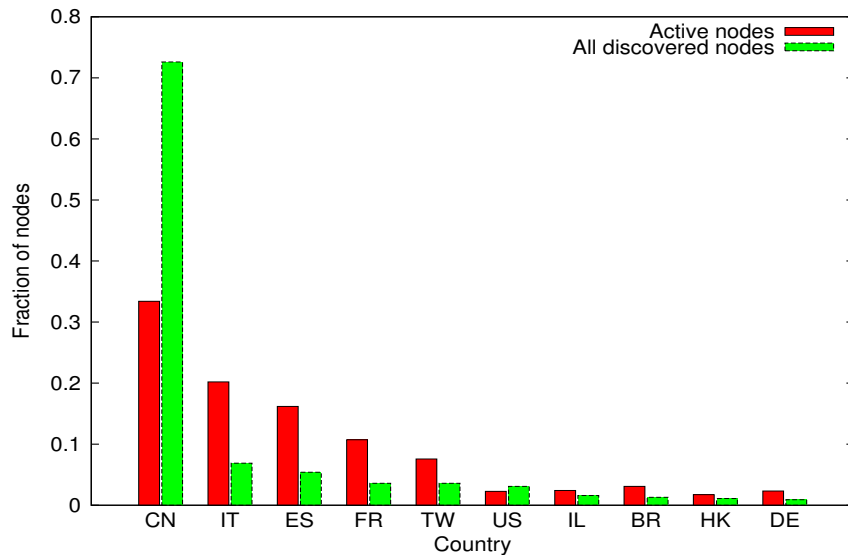


Figure 3.3: Geographical distributions (top ten countries) of both active nodes and all discovered nodes in KAD, as seen in an exemplary snapshot captured on 12.09.2013

An interesting observation in these results is the high disparity between the geographical distribution of active nodes and the geographical distribution of all discovered nodes (including both active nodes and stale routing table entries). The highest disparity is associated with the Chinese users: they represent about 73% of all contacts, but only about 33% of the active nodes. This disparity can be attributed to some KAD client software in China such as FlashGet and Thunder that generate a new random KID at the beginning of each new session [YFX⁺09].

3.4.2 Crawling Performance

We evaluate here the crawling performance of KadSpider, with respect to the preset design requirements (Section 3.2). In particular, we use two measures in our evaluation: (i) the crawling time and (ii) the completeness of the captured graph snapshots.

Crawling Time

The time that is required to perform a crawl consists of two interleaving time elements: (i) the time for discovering the nodes and (ii) the time for downloading their routing tables. The time for downloading a routing table depends on the maximum number of KAD_REQs needed to download a routing table, and it is also influenced by the aforementioned six-second restriction of KAD’s flooding protection mechanism.

During the full crawls, the average value of the first time element was 189.8 seconds, while the average value of the maximum number of sent KAD_REQs was 74. The worst case scenario is to discover the node with the maximum number of non-empty sub-trees at the end of the crawl. The crawling time in that case, using the aforementioned average values, is:

$$\text{Crawl time} = 189.8 + 6 \times 74 = 633.8 \text{ seconds.}$$

However, when we enabled the adaptive crawling rate technique in our experiments (Section 3.3.1), the average crawling time increased to 907.85 seconds (minimum = 518 ; maximum = 1311 ; median = 922 ; standard deviation = 151.29).

In order to measure the impact of the reported average crawling time on the accuracy of the captured graph snapshots, we performed the following experiment. Using KadSpider, we searched for 10000 online nodes and downloaded their routing tables. After 907.85 seconds⁵ from the beginning of the crawl, we queried the nodes for their routing table entries again. We repeated the experiment ten subsequent times and reported (i) the ratio of stable nodes and (ii) the ratio of stable routing table entries, both are calculated with respect to the previous crawl.

As shown in Figure 3.4, the results show that the changes in the discovered nodes or their routing tables, over the reported average crawling time, are very small in general: over the nine comparisons, the average stability ratios of online nodes and their routing table entries were 95.8% and 90.2%, respectively. Consequently, we consider the impact of the reported average crawling time on the accuracy of the captured graph to be small enough to maintain the properties of interest.

Completeness of the Captured Graph Snapshots

In theory, the completeness of captured snapshots is measured with reference to a ground truth. In our case, however, such knowledge (*i.e.* the real KAD graph) is unavailable. Instead, we measured the completeness of our snapshots as the ratio of received KAD_RESs to the sent KAD_REQs.

Using this measure, the average completeness values in our partial crawls and the full crawls were 97.53% and 84.48%, respectively. We summarize the main related statistics in Table 3.2. Note that the measured completeness represents a lower bound on the actual completeness. This is because it is likely that the crawling packets that are considered lost by the aforementioned completeness measure were learned via other KAD_RESs. More precisely, outgoing edges of the target node may returned in responses to crawling queries targeting neighbouring sub-trees (if the sub-trees contain less than β contacts). In this case, the queried

⁵ This is our reported average crawling time in the full crawls.

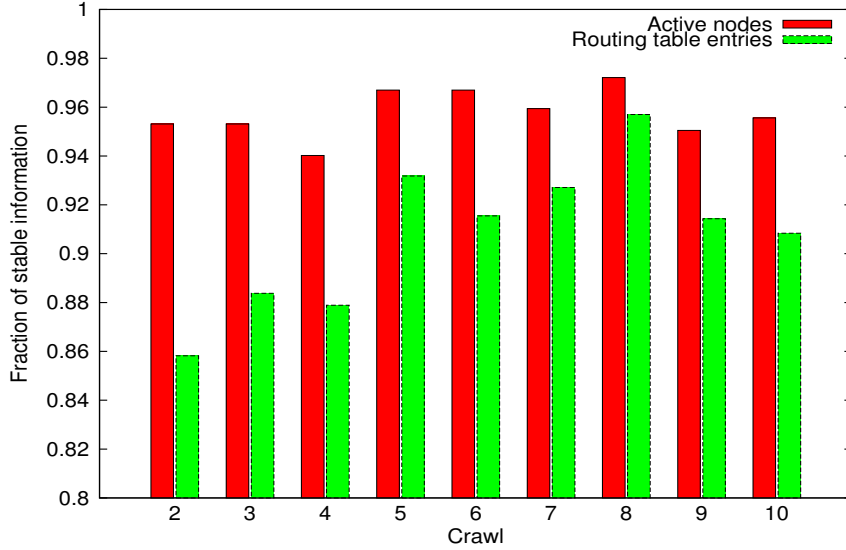


Figure 3.4: Stability ratios of 10000 online KAD nodes and their routing table entries after 907.85 seconds (our reported average crawling time for the full crawls)

node complements the β contacts from closest buckets to the targeted sub-tree. To validate this assumption, we compared the total number of routing table entries (*i.e.* with duplications) that are returned in KAD_RESs during an exemplary full crawl to the corresponding number of unique entries. The ratio was 1.89. That is, each routing table entry was downloaded nearly twice, on average. This result confirms our assumption that the actual completeness of captured graphs is higher than the completeness value reported by our aforementioned completeness measure.

Table 3.2: Completeness statistics of 65 KAD graph snapshots captured in the full crawls

	Full crawls (%)	Partial crawls (%)
Minimum	78	90
Maximum	93	100
Median	84	97
Average	84.48	97.53
Std. dev.	3.23	1.6

We attribute losses of crawling messages to two main factors: (i) using UDP-based crawling packets (UDP is unreliable) and (ii) the massive amount of crawling traffic (which causes network congestions). While we cannot do anything with the first factor (neither the UDP nature nor the type of crawling messages can be changed), we succeeded to mitigate the impact of the second factor by the above described design and deployment techniques (Section 3.3.1). Note

that the reported completeness can be improved by migrating our crawler to a more capable measurement environment. That is, by crawling each identifier space section multiple times in a synchronized way, message losses decreases exponentially to the degree of crawling parallelism.

3.5 Discussion

We summarize the performance results of KadSpider as follows: the reported average time to conduct a full crawl and the average completeness of the captured snapshots are 907.85 seconds and 84.48%, respectively. These values imply that KadSpider does not capture exact snapshots of the KAD graph. However, during the reported crawling time, 95.8% of the discovered online nodes and 90.2% of their routing tables' entries, on average, remained intact. That is to say, the impact of the crawling time on the accuracy of the captured snapshots is small. Furthermore, we explained why the actual completeness is expected to be better than the reported one. According to these results, we argue that the captured graph snapshots are close the original graph in a statistical sense. That is, the captured snapshots maintain the five properties that we aim to analyse in our study.

KadSpider cannot be directly compared to previous KAD crawlers for two reasons: First, the amount and the type of targeted information are different. More precisely, KadSpider captures graph snapshots, while other crawlers download partial information. Second, previous crawlers were not restricted by the recently added KAD's flooding protection mechanism.

KadSpider outperforms previous crawlers in terms of the time and the number of crawling packets for downloading nodes' routing tables. More precisely, KadSpider's technique of querying sub-trees of buckets enables it to target 2.5 buckets (download up to 25 contacts) per query, on average. In contrast, previous crawlers (except Rememj [YLX⁺11]) target one bucket only (*i.e.* download up to ten contacts) per query. That is to say, by this technique alone, KadSpider can download routing table buckets about 2.5 times faster, using about 2.5 times fewer queries. In addition, KadSpider's results become much better when it avoids querying empty buckets. As shown in Table 3.1, 74 queries were used at maximum by KadSpider to download an entire routing table. In contrast, previous KAD crawlers (except Rememj) can do that with 636 queries (the number of buckets in a perfect routing table [Bru06]). That is about 8.6 times the maximum number of queries used by KadSpider.

Regarding Rememj, it uses KAD's bootstrap packets to download up to 20 contacts by each crawling query. However, we argue that our crawling methodology outperforms Rememj for two reasons: First, bootstrap responses return random

contacts. Hence, Rememj cannot guarantee coverage of the entire routing table, while KadSpider can guarantee that using KAD's routing packets. Second, the number of contacts contained in a bootstrap response (up to 20) is smaller than the number of contacts that can be contained in a routing response (up to 31).

3.6 Summary

We presented in this chapter the design and evaluation of KadSpider, our own crawler for capturing representative graph snapshots of real Kademlia-type systems. Our results showed that KadSpider is capable of capturing representative snapshots of the KAD graph, and that it significantly outperforms prior crawlers in terms of the time and number of messages required to download entire routing tables. The results also showed that KAD is still a very active P2P system.

Table 3.3 summarizes the design elements and features that enable KadSpider to fulfil its preset design requirements (Section 3.2).

Table 3.3: KadSpider: Mapping design elements to the requirements

	R1	R2
Dividing the routing table tree into sub-trees of buckets	✓	✓
Avoiding querying empty buckets	✓	✓
Adaptive crawling rate	✓	✓
Distributed crawling architecture	✓	✓
Monitoring the crawling process		✓

In the next two chapters, we use the graph snapshots that we captured by KadSpider to analyse topological and routing table properties of KAD.

Topological and Routing Table Properties of KAD

We continue in this chapter the study that we started in Chapter 3 on characterizing topological and routing table properties of Kademlia-type systems. In particular, we present the first part of the characterization focusing on four properties: (i) the degree distribution, (ii) the completeness of routing information with respect to global knowledge of the system graph, (iii) the graph resilience in face of random node departures or removals, and (iv) the graph resilience in face of targeted node removals.

These properties represent predominant concerns when designing distributed systems with outstanding performance, reliability, and availability features. More precisely, the degree distribution describes the network structure in general, and it is necessary to calculate the resilience. Furthermore, the accurate analysis of nodes' degrees over the system graph can be utilized in the detection of attacks and other anomalies. Regarding the completeness of routing information, it directly relates to routing (*i.e.* lookup) performance. As for the analysis of the graph resilience both to random node departures and to intentional removals, it provides direct information about the quality of the system graph for communications. For instance, the resilience is of interest to determine whether a certain DHT is suitable for botnet command and control or not [DNF⁺08, DGLL07].

In contrast to former studies that relied on oversimplified simulations, we analyse the aforementioned properties based on (i) snapshots of real KAD graphs captured by KadSpider and (ii) synthetic graphs of KAD generated both by our own KAD simulator and by approximating standard models. In addition, we compare the results of the measured graphs and the synthetic graphs to each other, and we also identify the reasons for the observed differences.

4.1 Related Work

During the last years, a considerable amount of research work considered several aspects of P2P systems (*e.g.* see [HKLM03, TBF⁺03, CMSS13, PGES05, SR09, SR06b, SENB09, YFX⁺09, RSS15] and the references therein). However, very few studies considered system-wide topological and routing table properties. Existing studies are mainly concerned with protocol-specific analysis, whereas our goal lies in analysing these properties in both reality and models. Early work on characterization of P2P overlay topologies considered two unstructured P2P systems: Gnutella [SRS08, MIF02, JAB01] and Kaaza [LKR04]. However, their characterizations are not representative for structured P2P systems like Kademlia.

To the best of our knowledge, the only three papers that analysed global topological properties of structured P2P systems are [HYEN09, DNF⁺08, DGLL07]. They studied the effectiveness of P2P-based botnets in comparison to other botnet structures. In more detail, Ha et al. [HYEN09] studied several graph-theoretic properties of KAD, with the goal to analyse the effectiveness of structural detection of KAD-based botnets, via monitoring of overlay traffic. Their key conclusion was that structural detection of KAD-based botnets is not effective. Davis et al. [DNF⁺08] analysed the resilience and other characteristics of Overnet (a Kademlia-type system) and compared its properties with Gnutella and two theoretical graph models. Their results showed that Overnet is less resilient to failures than the three other graphs, but it surpasses them in terms of resilience in face of targeted attacks. Dagon et al. [DGLL07] provided a taxonomy and a detailed analysis of P2P-based botnets, in which they approximated graphs of structured P2P-based botnets by the Erdős-Rényi's model of random graphs [ER59].

The three aforementioned studies are based on small-scale simulations, disregarded churn and had some other oversimplified assumptions. Such limitations lead to mischaracterizing the degree distribution and graph resilience. Instead, we use both real-world measurements and simulations, and we also compare them to each other. In addition, none of the studies above analysed the completeness of routing tables, an important routing performance factor.

The only comprehensive measurement study in a similar area is the characterization of routing tables in KAD by Stutzbach and Rejaie [SR06a]. Though their focus was on the aliveness and completeness of routing tables, they indirectly sampled the out-degree distribution. Their analysis, however, is based on a small sample of routing table buckets. Hence, their work cannot capture the in-degree, the resilience, nor the completeness with respect to global graph knowledge. Capturing these properties requires the knowledge of the system graph (*i.e.* entire routing tables of all active nodes).

4.2 Background

4.2.1 Topological and Routing Table Properties

In the following, we use Definition 1 when talking about the KAD graph. We denote nodes by u and v , and we denote the expected value of a random variable X by $\mathbb{E}(X)$.

Degree Distribution

The in-degree of a node v in a graph is $d_-(v) = |\{(u, v) \in E : u \in V\}|$, *i.e.* the number of incoming edges. Correspondingly, the out-degree $d_+(v) = |\{(v, u) \in E : u \in V\}|$ is the number of outgoing edges of v (*i.e.* the sum of entries in v 's routing table buckets). The degree $d(v) = d_-(v) + d_+(v)$ is the sum of in- and out-degree. The (in-/out-)degree distribution D of a graph G maps each non-negative integer k to the fraction of nodes in G that have (in-/out-)degree k .

Routing Table Completeness

We define the maximal out-degree $M_v(D)$ to be the number of contacts that can be contained in the routing table of a node v if v has global knowledge of all graph vertices. We compute the routing table completeness as the ratio $r = d_+(v)/M_v(D)$. This measure is an indicator for the impact of missing knowledge of contacts and outdated information (*i.e.* stale routing table entries) on the completeness of routing tables.

In addition, we compute the expected maximal out-degree $\mathbb{E}_v(D)$. More precisely, let $X_{n,p} \sim B(n-1, p)$ be a binomially distributed random variable with n trials and success probability p . Let n denote the number of nodes, and let p denote the fraction of contacts a bucket is responsible for. The expected number of entries $\mathbb{E}(C_{n,p})$ in the bucket is:

$$\mathbb{E}(C_{n,p}) = \sum_{i=1}^{10} \left(1 - \sum_{j=0}^{i-1} \binom{n-1}{j} p^j (1-p)^{n-1-j} \right).$$

Given that a perfect KAD's routing table contains eleven buckets in the fourth level and five buckets in each of the levels 5 to 127 (Section 2.3), $\mathbb{E}(D)$ is given by:

$$\mathbb{E}(D) = 11\mathbb{E}(C_{n,0.5^4}) + 5 \sum_{i=5}^{127} \mathbb{E}(C_{n,0.5^i}). \quad (4.1)$$

Accordingly, the expected completeness ratio: $r_e = d_+(v)/\mathbb{E}_v(D)$.

Graph Resilience

We define the resilience of a graph, with regard to a removal strategy R , to be the fraction of nodes (including their incident edges) that need to be removed so that no giant connected component remains. At this point, the graph breaks into small partitions, each containing less than half of the remaining nodes. In agreement with similar studies (*e.g.* [DNF⁺08]), we consider random removal (or departure) of nodes as a model for failures, and removal of the nodes with the highest (in-/out-)degree as a model for targeted attacks.

In general, there are two main approaches to compute the resilience of a graph. The first is to simulate the resilience by iteratively removing nodes from the graph and then recomputing the giant component. We did not use this approach since it is computationally expensive and does not scale to graphs as large as the measured KAD graphs (on average, each snapshot included about 452,740 concurrent active nodes and about 3,812,030 unique routing table entries).

The second approach to compute the resilience [CEBAH00,CEBAH01] uses the degree distribution of the graph's nodes. This approach requires a computational overhead of $\mathcal{O}(\min\{m_+m_-, n\})$, where $m_{-/+} = \max_{v \in V} d_{-,+}(v)$ is the maximum in- and out-degree. This approach scales to the size of the real KAD graph, and therefore we decided to use it in our analysis. However, this approach is designed for bidirectional graphs, while the KAD graph is unidirectional. Therefore, we first need to adapt it for unidirectional graphs before adopting it in our analysis. Towards this end, we used the adaptation method from [SRS14a], which can be summarized as follows:

For unidirectional graphs with a give degree distribution D (modelled as a random vector $D = (D_-, D_+)$), G has a giant connected component almost surely if

$$\mathbb{E}(D_-, D_+) - \mathbb{E}(D_+) > 0 ,$$

and G almost surely does not have a giant connected component if the above quantity is less than 0 [MR95,NSW01].

In order to obtain the resilience under a removal strategy R , the degree distributions $\tilde{D}^p = (\tilde{D}_-^p, \tilde{D}_+^p)$ after removing a fraction p of nodes are derived. Then, we determine the fraction p that needs to be removed such that no giant component exists as

$$\mathbb{E}(\tilde{D}_-^p, \tilde{D}_+^p) - \mathbb{E}(\tilde{D}_+^p) = 0 . \quad (4.2)$$

For random failures, Eq. 4.2 resolves to

$$(1 - p)\mathbb{E}(D_-, D_+) - \mathbb{E}(D_+) = 0 . \quad (4.3)$$

For targeted attacks, let $r(p, k, j)$ be the fraction of removed nodes with in-degree k and out-degree j , and also let $\bar{F}_D(d)$ be the probability of a node to have a degree $\geq d$. For a fraction p of removed nodes with the highest degree:

$$r(p, k, j) = \begin{cases} P(D = (k, j)), & \bar{F}_D(k + j) \leq p \\ 0, & \bar{F}_D(k + j + 1) \geq p \\ \frac{p - \bar{F}_D(k + j + 1)}{\bar{F}_D(k + j + 1) - \bar{F}_D(k + j)}, & \text{otherwise.} \end{cases}$$

The fractions of removed outgoing edges, p_- , is then given by

$$p_- = \sum_{k=1}^{m_-} \sum_{j=1}^{m_+} r(p, k, j) \frac{k}{E(D_-)},$$

and similarly for p_+ . Eq. 4.2 can be then transformed to

$$(1 - p_-)(1 - p_+)(\mathbb{E}(D_-, D_+) - \mathbb{E}(D_+) - 2kjr(p, k, j)) = 0 \quad (4.4)$$

so that p can be computed from the definitions of p_- and p_+ and Eq. 4.4.

4.2.2 Standard Graph Models

In addition to KAD, we deal in this study with two standard graph models: (i) Erdős-Rényi's random graph model [ER59] and (ii) Barabási-Albert's scale-free model [BA99]. We give an overview of them in this section.

Erdős-Rényi's Random Graph Model

The Erdős-Rényi's (ER) model [ER59] generates random graphs with two parameters: (i) $|V|$, the number of vertices, and (ii) P , the probability of an edge to exist. The graphs have binomial degree distributions, in which each node is connected with an equal probability to the other $|V| - 1$ nodes, resulting in $|E| = \binom{|V|}{2}P$ edges, on average. Graphs of Kademlia-type systems were approximated by this model [DGLL07].

Barabási-Albert's Scale-free Model

The Barabási-Albert's (BA) model [BA99] generates graphs in which the probability that a node connects with k other nodes is roughly proportional to $k^{-\gamma}$, for some constant γ . This results in a power-law degree distribution: few nodes will have very high degrees, while most the nodes have very low degrees. This distribution is also called scale-free distribution. Such a distribution is widely observed in several systems, such as the Internet and some P2P and social networks. It has been shown that BA's scale-free graphs are highly resilient in

face of failures [CEBAH00] but not in face of targeted attacks [CEBAH01]. That is, removing the highly connected nodes from these graphs may easily impact their connectivity.

4.3 Methodology

As we mentioned earlier, our analysis is based on both measured graphs and synthetic graphs. We give in this section an overview of the analysed graphs and our methodology to capture them. As illustrated in Figure 4.1, we extract two graphs from each measured and each synthetic KAD graph: the first graph (hereafter, active graph) includes only online nodes, while the second graph (hereafter, complete graph) includes all nodes (both active nodes and stale routing table entries). For the measured graphs, we extract two other graphs from each of the complete and active graphs: one includes nodes that use the same IP address and UDP port but multiple KIDs (due to manipulation of the client software), while the other graph excludes them. We analyse each of the four graphs separately.

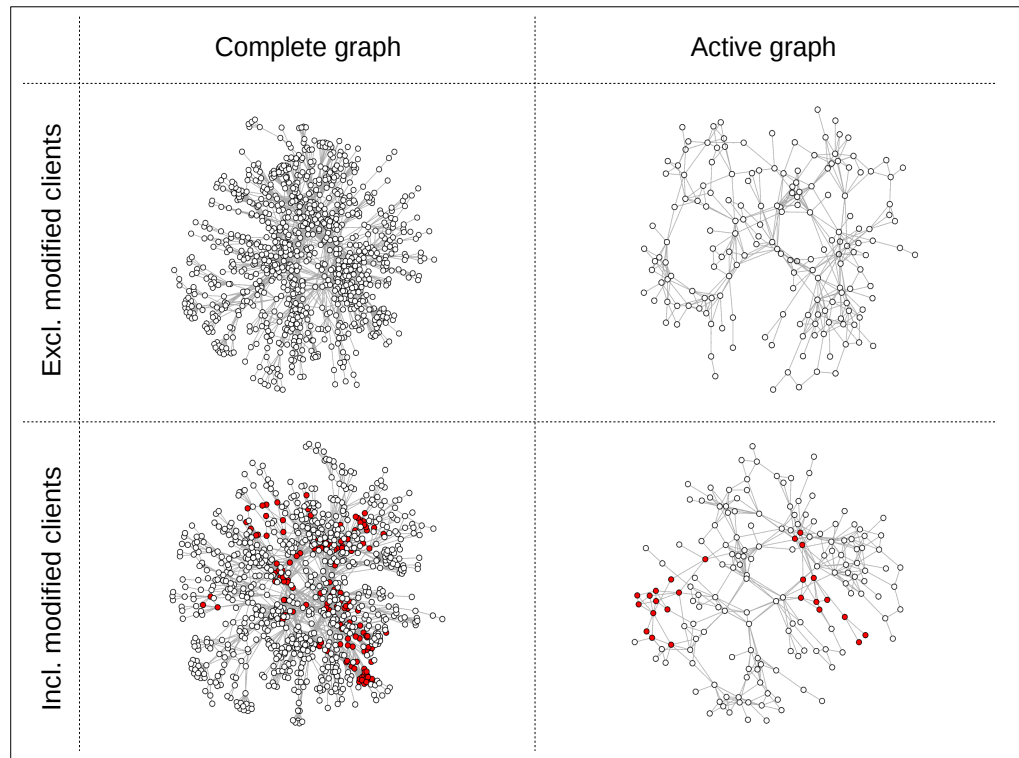


Figure 4.1: Illustration for the four graphs extracted from a graph snapshot of KAD. The red-coloured nodes in the two lower graphs denote modified clients.

4.3.1 Measurements

Our analysis of measured KAD graphs is based on our dataset of KAD graph snapshots which we collected using KadSpider (Section 3.4.1). The dataset consists of 65 graph snapshots with 452,740.35 concurrent active nodes and 3,812,030.43 unique routing table entries, on average. The analysis results that we obtained from all measured graphs are very similar and lead to the same conclusions. Therefore, we select an exemplary graph snapshot and show its results here to represent the entire dataset. In particular, we consider a graph snapshot captured on 12.09.2013 (started at 20:00 GMT). This snapshot includes 498,765 active nodes and 4,194,667 unique routing table entries. These values imply that about 88% of routing table entries in this snapshot are stale. The snapshot also contains 65,623 nodes (about 13.2% of the active nodes) using multiple identifiers, mainly due to software manipulation.

4.3.2 Simulations

KAD Graphs

In order to obtain synthetic KAD graphs comparable to the measured graphs, we need to simulate KAD as it is implemented in the real client software. Existing P2P simulators (such as PeerSim [MJ09], PeerfactSim.KOM [SGR⁺11], and OverSim [BHK07]) implement different structured P2P systems, including Kademlia-type ones, but none of them implements KAD. Although KAD is derived from Kademlia, there are several differences between the two protocols which make developing a new simulation model for KAD more straightforward than adapting an existing Kademlia-type model.¹

Towards this end, we developed a new simulation model of KAD. In particular, we implemented KAD as a new P2P protocol in OverSim, a widely used simulator which is based on the OMNeT++ framework [OMN]. Our simulation model implements the protocol design and settings as in the eMule 0.50a client software.

We performed simulations on three graph sizes: 1000, 5000, and 10000 nodes. We performed 20 simulation runs for each of the three graph sizes, and the results for the same size were almost identical. The results that we explain in Section 4.4 represent three synthetic graphs (one for each of the three graph sizes). The simulator assigned each node a 128-bit identifier generated uniformly at random. Each simulation run lasted 12000 simulation seconds, during which each online node requested a random identifier once every 300 simulation seconds. The snapshots were captured after the target size is reached and the nodes have populated their routing table buckets with contacts.

¹ For a detailed comparison between Kademlia and KAD, the reader is referred to [Mys06].

Regarding churn, our simulations applied the settings of [BH12], which were believed to resemble nodes' behaviour in the real system [SR06b, SENB09]. In particular, we assigned the nodes a Weibull distributed session times with shape $k = 0.5$, and average life time and average dead time both of 10000 seconds.² With these settings, the resulted ratio of stale contacts in the complete graphs were about 45%, 42%, and 30% in graphs containing 1000, 5000, and 10000 nodes, respectively.

Erdős-Rényi's Random Graphs

We used GTNA [SS13b] (Graph-Theoretic Network Analyser, a general framework for graph and routing analysis) to produce ER's random graphs. In particular, we generated static ER's random graphs containing 7000 nodes each³. We chose this size to make the ER's random graphs comparable with our 10000-node KAD graphs in which 30% of the nodes are stale. The generated ER's random graphs have average degrees very similar to the respective KAD graphs.

4.4 Results

We now discuss our analysis results: the degree distribution in Section 4.4.1, the routing table completeness in Section 4.4.2, and the graph resilience in Section 4.4.3. We used GTNA to compute the degree distributions and graph resilience, as they are defined in Section 4.2.1.

4.4.1 Degree Distribution

In this section, we first describe our expectations for the degree distributions. After that, we give an overview of the degree distributions of the measured graphs, and we also analyse the impact of stale routing table entries and manipulated clients on the degree distribution. Next, we compare the measured KAD graphs to synthetic graphs both of KAD and ER's random networks.

Expectations

According to the description of Kademlia [MM02] and the implementation of KAD in eMule [emu], we have the following expectations for the degree distribution of the KAD graph:

² For more information about the churn models that are implemented in OverSim, the reader is referred to: <http://oversim.org/wiki/OverSimChurn>.

³ Note that without churn, both the active graph and the complete graph are equivalent.

- The out-degree is bounded by the maximal number of routing table entries. Hence, we expect many nodes with similar out-degrees and none with a very high out-degree. The in-degree, in contrast, is unbounded because stable nodes attract more and more links over time. We thus expect a power-law in-degree distribution.
- We expect the degree distributions of the active graph and the complete graph to be highly different, given the large fraction of stale routing table entries in the complete graph (88%). The complete graph is bound to have a lower average degree and more nodes with a low degree corresponding to stale routing table entries. However, the complete graph is expected to have a higher maximal degree, because all entries in the routing tables of active nodes are counted.
- The nodes that use multiple identifiers potentially have higher degrees than the degrees of ordinary nodes, because they are expected to have neighbours from a larger fraction of the identifier space. However, we expect their influence on the degree distribution to be marginal, because they only represent a small fraction of the graph size.
- The synthetic KAD graphs should have degree distributions relatively similar in shape to the degree distributions of the measured graphs, although shifted due the lower number of nodes in the simulations.
- We expect the differences between the degrees of highly-connected nodes and low-connected nodes in the degree distribution of KAD graphs to be higher than in the binomial degree distribution of the ER model. In particular, the ER model does not result in nodes with very high in-degrees like long-aged KAD nodes.

Analysis Results

We discuss here the results that we obtained from analysing degree distributions of our measured and synthetic graphs. We first give an overview of the distribution shapes of the measured graphs. After that, we discuss the impact of stale routing table entries and modified clients. Next, we compare the distributions of the measured graphs to those of the synthetic graphs. To a high extent, the actual results agree with the expectations above.

Overview: Figure 4.2 depicts the distribution shapes of the four graphs that we extracted from our exemplary measured graph. The out-degree distribution (Figure 4.2a) increases up to a mode, and drops rapidly afterwards when the capacity of the routing table is reached. Nodes with low out-degrees are the nodes

that did not fill their routing tables yet, *i.e.* nodes that are still in the connection or bootstrapping phase.

The in-degree distribution of the complete graph (Figure 4.2b) has the signature of power-law distribution (looks like a line on a log-log plot) with a spike around the average in-degree. The same observations apply for the active graph except that the nodes with in-degrees smaller than the average value have a slowly decreasing (nearly constant) distribution.

The shapes of degree distributions (Figure 4.2c) are asymptotically equal to the respective in-degree distributions, but they also show the mode of the out-degree distributions at degree of roughly 1000.

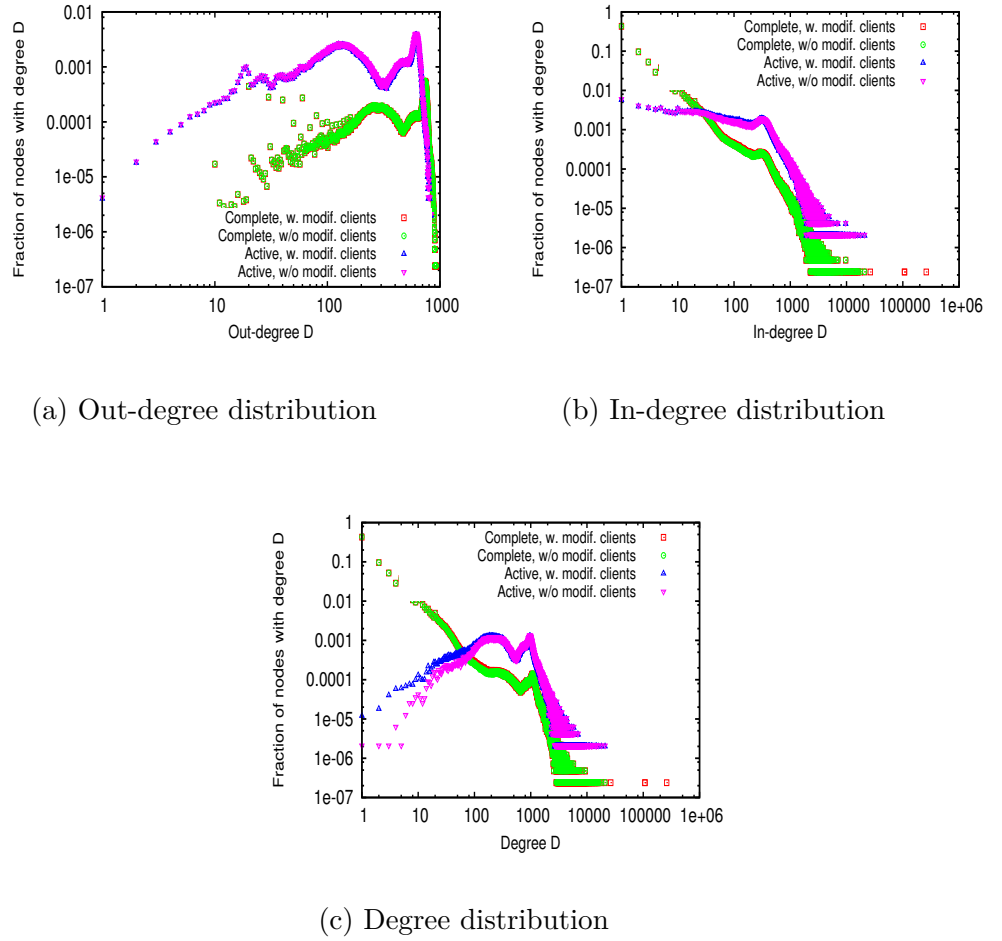


Figure 4.2: Degree distributions of measured KAD graphs

Impact of stale entries and modified clients: In Figure 4.2, we also can observe (i) the small impact of manipulated clients and (ii) the drastic impact of stale routing table entries. In particular, nodes having multiple identifiers do not

change the general shape of the curves but lead to higher degrees, as expected. In particular, these are the nodes with the highest in-degrees.

Regarding the impact of stale routing table entries, we can see that the out-degree distribution when considering only active nodes shows higher probabilities up to a routing table size of about 1000. However, the out-degree distribution considering all routing tables' entries (*i.e.* in the complete graph) shows a higher probability for large degrees. Note that when including all entries, 88% of nodes are with out-degree of zero (inactive nodes or stale routing table entries). For the in-degree distribution, on the other hand, the complete graph shows a considerably higher fraction of nodes with a low in-degree, which corresponds to the high fraction of stale routing table entries. As discussed above, the active graph has much fewer nodes with low in-degrees to form a standard power-law in-degree distribution. Consequently, such a more uniform in-degree distribution is very likely to make the active graph more resilient to targeted attacks than both standard scale-free graphs and the complete KAD graph, because it reduces the graph reliance on highly connected nodes (also called hubs).

Measured graphs versus synthetic graphs: We now compare the degree distributions of the measured graph with those of the synthetic graphs. We exclude measured nodes with multiple identifiers from the comparison, because they do not appear in the synthetic graphs. We plot the comparison results both for the complete graphs (Figure 4.3) as well as for the active graphs (Figure 4.4).

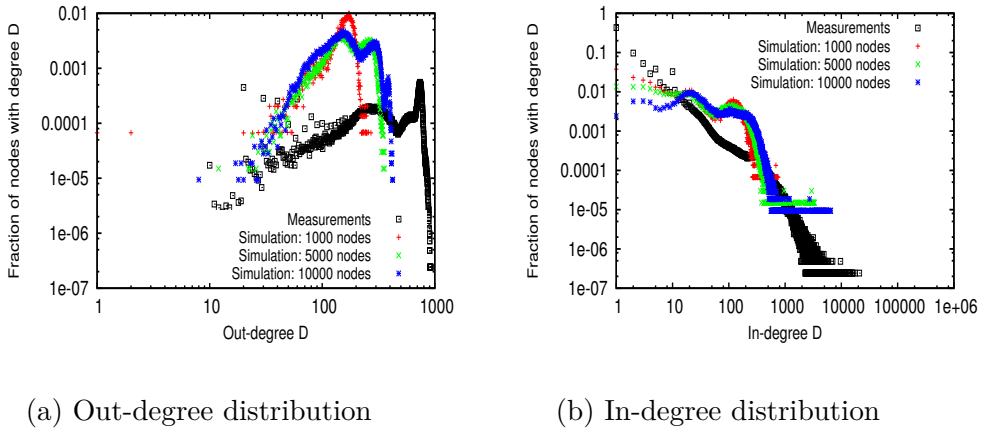


Figure 4.3: Degree distributions of complete KAD graphs (log-log scale): measurements vs. simulations

In the complete graphs, the out-degree distributions (Figure 4.3a), both of the measured and the synthetic KAD graphs, show the characteristic drop at the

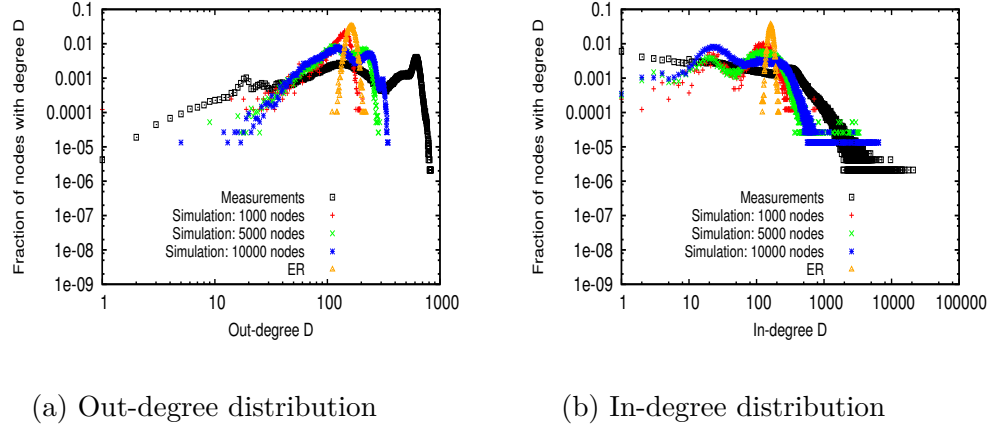


Figure 4.4: Degree distributions of active KAD graphs (log-log scale): measurements vs. simulations

maximal routing table size. However, there is a difference in the shapes with regard to the in-degree distributions (Figure 4.3b): while the in-degree distribution of the measured graphs exhibits a constant negative slope, the synthetic graphs result in local extrema. These observations also apply for the active graphs (Figure 4.4).

We show in Figure 4.4 the out- and in-degree distributions of an ER's random graph with 7000 nodes. Although this graph has a similar size, average degree, and staleness degree as the active part of the 10000-node KAD graph, the degree distributions of the two graphs are different. In comparison to the above explained distribution of the synthetic KAD graph, the random graph has a binomial degree distribution narrowed around a degree $k = P(|V| - 1)$ (which is to be expected for the ER model). Nodes with very small or large (in-/out-)degrees do not exist in the ER's random graph.

4.4.2 Routing Table Completeness

DHTs, in theory, achieve relatively good routing performance: typically $\mathcal{O}(\log|V|)$ routing steps. These analytical derivations, however, disregard churn and its consequences, namely the staleness and incompleteness of routing information. We showed in Section 3.4.1 that churn results in a staleness ratio of about 88% in real KAD routing tables. Later, we showed in Section 4.4.1 that such high ratio has a drastic impact on the topology. Hence, it is expected to have a negative impact both on routing performance [SR06a] and graph resilience.

In this section, we extend our analysis for KAD, considering a previously disregarded routing performance factor: the completeness of routing tables with respect to global graph knowledge.

Expectations

Before discussing the actual completeness results, we first compare the values of the actual average out-degree (out-deg) with the expected maximal out-degree ($\mathbb{E}(D)$) assuming global graph knowledge (computed according to Eq. 4.1). After that, we compute the ratio r_e between out-deg and $\mathbb{E}(D)$. Table 4.1 displays the results, which confirm our expectations for degree distributions. In particular, the impact of the high stale entry rate in the measured graphs on the average out-degree is high: the average out-degree of about 56 in the complete measured graph (of roughly four million nodes) is below the average out-degree of about 87 that is observed in synthetic graphs of only 1000 nodes. When only considering active nodes, the average degree is higher in measurements than in simulations, as expected, given the larger graph size.

Table 4.1: Average out-degree, expected out-degree, and the expected completeness ratio r_e : measured vs. synthetic KAD graphs

Graph	Complete			Active		
	out-deg	$\mathbb{E}(D)$	r_e	out-deg	$\mathbb{E}(D)$	r_e
Sim-1000	87.02	285.7	0.305	129.45	271.11	0.477
Sim-5000	112.00	401.86	0.279	155.13	382.85	0.405
Sim-10000	141.00	451.87	0.314	162.81	431.04	0.378
Measured	56.3	887.45	0.063	363.76	733.86	0.496

Similar conclusions hold for the ratio r_e . In the synthetic graphs, r_e is between 0.279 and 0.314, about five times the corresponding value in the measured graphs when considering all routing table entries. However, in the active graphs, the ratios both in the measured graphs and in the synthetic graphs are close to each other. More precisely, routing tables in standard KAD are expected to include almost half the active contacts that they would include if they know the system graph. We compare now the expected values with the actual ones.

Actual Results

We show now the results of the actual routing table completeness in KAD (based on the exemplary measured graph). In particular, we analyse the impact of stale routing information in three cases: (i) complete-1 includes the discovered active nodes and their routing tables (including the stale entries), (ii) complete-2 corresponds to the complete graph, and (iii) active corresponds to the active graph.

Figure 4.5 depicts the cumulative distribution function (CDF) of the measured routing table completeness. In general, the results show that nodes in KAD have

low to intermediate routing table completeness. In particular, 50% of the active nodes have completeness degrees of less than 34% and 44% in complete-1 and active, respectively. In the case of complete-2, the results are much worse since about 88% of the nodes already have out-degree of zero. We summarise the main related statistics in Table 4.2.

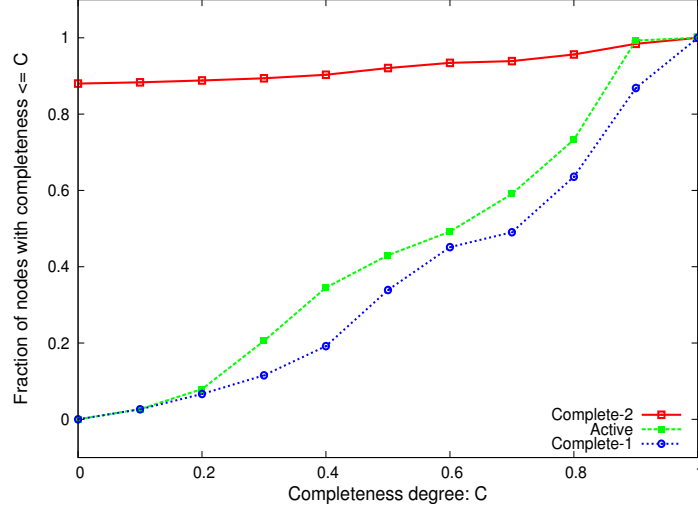


Figure 4.5: CDF of the actual routing table completeness of an exemplary measured KAD graph

Table 4.2: Routing table completeness statistics of a measured KAD graph

	Complete-1	Complete-2	Active
Minimum	0.9980	0.9980	0.9740
Maximum	0.0180	0	0.0110
Median	0.7560	0	0.6101
Average	0.6567	0.0837	0.5574
Std. dev.	0.2558	0.2373	0.2475

Comparing the average values of the actual completeness to the corresponding expected values (Table 4.1), considering the complete-2 and active cases, we can see that our expectations (according to Eq. 4.1) are somehow close to the actual values. The differences among the three cases confirm the impact of stale routing information on the completeness results, too.

In addition to the completeness itself, we analysed which routing table entries are missing with regard to their distances from the owners of the analysed routing tables. Figure 4.6 shows the distribution over the common prefix length of the node identifier and the missing entries in its routing table, considering the complete-1 case.

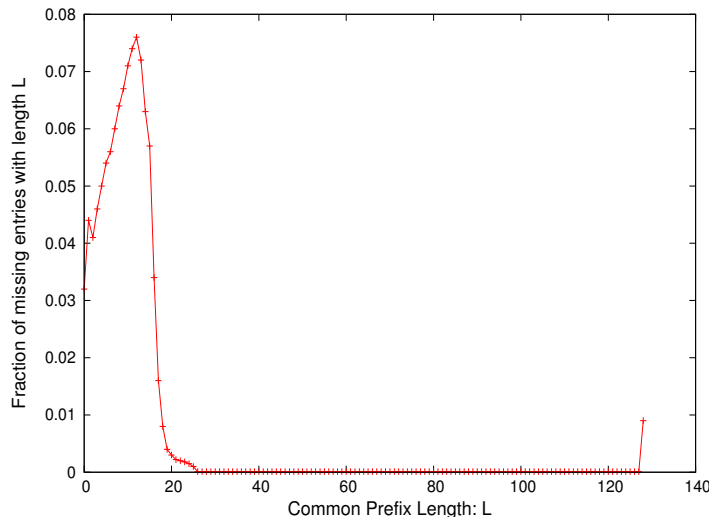


Figure 4.6: Distribution of missing routing table entries with respect to the common prefix length (representing routing tables of an exemplary measured KAD graph)

It can be seen that the number of missing entries with a low common prefix length is small, because there are many contacts fulfilling this criterion, and therefore it is expected to fill the respective buckets. The probability to miss an entry increases up to a common prefix length of 12 bits. After that, this probability decreases, mainly because the number of contacts with longer common prefixes is low (*i.e.* the buckets cannot be filled anymore). More particularly, there are very few contacts with a common prefix length between 25 and 127 bits, and consequently there are few missing entries in that range. However, there are missing entries with identical identifiers (*i.e.* with a common prefix length of 128) which mostly correspond to modified clients.

4.4.3 Graph Resilience

We analysed in Section 4.4.1 the degree distribution which has an impact on the system performance, but it does not directly provide information about the quality of the system graph for communications. This information can be obtained through analysing the graph resilience, a predominant concern when designing distributed systems, especially for highly dynamic systems.

We analyse in this section the resilience of KAD graphs to failures (*i.e.* random node removals or departures), as well as to targeted attacks. We compare the results of measured KAD graphs with both synthetic KAD graphs and ER's random graphs. We also contrast our findings with results of prior studies. The resilience computations are performed according to Eq. 4.3 and Eq. 4.4.

Expectations

Inspired by the analysis results of degree distributions in KAD (Section 4.4.1), we have the following expectations for the graph resilience:

- The graph resilience to failures is high and increases with the graph size, as shown for standard scale-free graphs in [CEBAH00]. Therefore, we expect the measured graphs to be more resilient to failures than the synthetic graphs. In addition, the difference between the resilience of the active graph and the resilience of the complete graph to failures is likely to be small.
- We expect the KAD graph to be more resilient to targeted attacks than what was showed in [CEBAH01] for standard scale-free graphs. This is because the theoretical analysis in [CEBAH01] is only asymptotic, and our graphs have a very high average degree in comparison to their analysed graphs. We still expect the graphs to be more resilient to failures than to targeted attacks.
- We expect the active graph to be more resilient to targeted attacks than the complete graph, because the disparity between nodes with low degrees and nodes with high degrees is less pronounced in the active graph. That is, the hubs are less influential because they have less connections. Similarly, since this disparity is less pronounced in the ER's random graph than KAD graphs with the same size and similar average degree, we expect the ER's random graph to be more resilient in face of targeted attacks than the receptive KAD graphs.
- The synthetic graphs are expected to be more resilient to targeted attacks, because their maximal degree is lower. Hence, the impact of the removed hubs is much lower. As for the manipulated clients in the measured graphs, they tend to show higher degrees. Hence, they are expected to reduce the graph resilience to targeted attacks.

Analysis Results

Overview: Table 4.3 shows the resilience results both of the measured and the synthetic graphs, separately considering all nodes and only active nodes. The provided values characterize the fraction of nodes that need to be removed until there is no giant connected component.

The results confirm our expectations: on the one hand, the resilience to failures is well above 99% in all considered graphs. In addition, the resilience slightly improves with the graph size, being at its maximum in the measured graphs. The difference between the complete graph and the active graph is negligible in the

Table 4.3: Resilience values of measured and synthetic KAD graphs

Graph	Complete		Active	
	Random	Targeted	Random	Targeted
KAD-Sim-1000 (active: 553)	0.9928	0.5428	0.9929	0.9194
KAD-Sim-5000 (active: 2906)	0.9944	0.5776	0.9944	0.9119
KAD-Sim-10000 (active: 7004)	0.9951	0.7359	0.9951	0.8860
ER-7000	0.9950	0.9919	0.9950	0.9919
KAD-Meas: w/o modif. clients	0.9984	0.1100	0.9980	0.8625
KAD-Meas: w. modif. clients	0.9984	0.1105	0.9980	0.8443

case of failures. On the other hand, the active graph under targeted attacks is much more resilient due to the lower impact of hubs, as discussed above.

Impact of size, stale entries, and modified clients: Deriving the influence of the graph size is complex. Particularly, when considering all nodes, the increase of the size correlates with increased resilience in the synthetic graphs. Here, the increased average degree seems to be the dominating factor. Naturally, the resilience to targeted attacks seems very low in the case of the measured graphs. However, given that about 88% of the entries are stale (*i.e.* needing to remove about 12% of the nodes), it means that the majority of active nodes has to be removed.

When considering only active nodes, however, the resilience in general is higher, but decreases with the graph size. This is because the average degree is already high, so that the stronger proportional increase of the maximal degree is dominating. As a consequence, high-degree nodes become more important when the graph is larger, *i.e.* removing those nodes destroys the structure faster. Including nodes with multiple identifiers reduces the resilience of the measured graphs to targeted removals, as expected. Nevertheless, the difference is small (a reduction of about 2% only), which is to be expected because they represent only a relatively small fraction of the graph size.

KAD graphs versus ER’s model graphs: Comparing the results of the active synthetic KAD graph with the generated ER’s random graph (both have the same size and similar average degrees), we can see that both graphs have almost identical resilience in face of random departures. However, the resilience to targeted attacks is lower in KAD by about 11%. This is mainly because the difference between the degrees of highly-connected nodes and the degrees of low-connected ones is much higher in KAD than in ER’s random graphs. These results indicate that approximating graphs of Kademia-type systems by ER’s random graph is not accurate.

We also compared our results with the findings of Davis et al. [DNF⁺08]. Although the authors used a different measure of resilience, the conclusions derived from our and their graph resilience results should be comparable. In particular, using synthetic graphs of Overnet (a Kademlia-type system) and comparing them to ER's random graphs of the same size and similar average degree, they conclude that Overnet is less resilient to failures but more resilient in face of targeted attacks than ER's random graphs. We argue that their results mischaracterize Overnet properties, for two reasons: First, they are based on simulations that disregarded user dynamics and misbehaviour. Second, the simulations limited the node degree in a 25000-node Overnet graph to 20 edges, which is highly different from the degrees in the real system (Figure 4.4).

Conclusion: To sum up, we saw that deriving the resilience of large graphs by analysing the protocol on a small scale provides only qualitative information, and that the resilience increases (or decreases) according to the graph size. That is to say, the tendencies observed in the synthetic graphs are indicators for the resilience of the measured graphs. This also applies for approximating the KAD graph by ER's random graphs. However, the resilience of the measured graphs when considering both active and stale nodes (*i.e.* complete graphs) can be much lower against targeted attacks. In addition, anomalies such as nodes with multiple identifiers harm the resilience of the graphs to attacks (according to their population), and they were never simulated. These findings confirm that topological properties can be used to automatically detect anomalies and attacks.

Furthermore, our results regarding the resilience of measured KAD graphs are of interest on their own, regardless of their relation to synthetic graphs. They indicate that the KAD graph is highly resilient both to random failures and to targeted attacks. Hence, KAD is a suitable choice for distributed applications, such as critical infrastructures or botnets' command and control, requiring high resilience.

4.5 Summary

We analysed in this chapter topological and routing table properties of KAD. In particular, we analysed representative graphs collected from a large-scale measurement campaign, and compared them both with synthetic graphs generated by a novel simulation model of KAD as well as with ER's random graphs.

At a high level, the results showed that the rate of stale routing information significantly influences the average degree, the shapes of degree distributions, and the graph resilience to breakdowns. These properties also are influenced, relatively slightly, by the system size and presence of manipulated clients.

In more detail, we saw that the active KAD graph is highly robust not only to

random departure but also to targeted attacks, making it suitable for distributed applications requiring high resilience. Resilience to random departure and shapes of degree distributions are well modelled by KAD simulations. However, due to a greatly increased ratio of stale routing information, the complete graph in the real system is much more vulnerable to targeted attacks in comparison to estimations based on simulation results. We also showed that prior results that are based on oversimplified assumptions are not representative for the real system, in several cases. In addition, the analysis of routing information completeness showed that nodes in KAD achieve about half the maximal completeness value only, when considering active nodes. This result suggests that the standard approaches for discovering neighbours should be modified so that the nodes can increase the number of active contacts in their routing tables (to improve the routing performance).

We extend our characterization of KAD in the next chapter by studying a previously disregarded routing table property, which affects the routing performance: the diversity of node's contacts over the corresponding identifier space sections. Consequently, we propose and evaluate a lightweight modification for the original neighbour selection scheme, with the goal to improve the routing performance.

Improving the Routing Performance in Kademlia-type Systems

The routing (or lookup) protocol of Kademlia-type systems received high attention from the research community in the last years. Several studies (*e.g.* [LSG⁺05, SR06a, SCB10, FPJ⁺07, CW07, JOK11, RSS15]) identified limitations in the standard routing protocol, and consequently raised doubts about its suitability for latency-sensitive applications. In addition, part of these studies proposed modifications to the standard protocol design, aiming to improve the routing performance. Considered performance metrics in these studies are the hop count, the latency, and the signalling overhead (*i.e.* the number of messages sent during a lookup).

We aim to improve the routing performance of Kademlia-type systems to maintain high service quality in case of large number of participants. This is particularly important when these systems are adopted by latency-sensitive applications such as P2P video streaming [Jim13]. Towards this end, we study a previously disregarded routing performance factor: the diversity of neighbours' identifiers inside each routing table bucket over the corresponding identifier space sections.

We show that maximizing this diversity can improve the routing performance. Consequently, we propose a new neighbour selection scheme that attempts to maximize this diversity, aiming to reduce the lookup's hop count. The scheme is compatible not only with the standard Kademlia system and its variations but also with previous improvements. Moreover, the scheme does not change the standard routing protocol nor the routing table structure, and it does not cause signalling overhead. It is also consistent with the original policy that gives preference for long-lived contacts and hence maintains the logic behind it (Section 2.1). It requires only small changes to the routing table's maintenance processes (Section 2.3.1), causing small overhead for computing diversity degrees.

In order to get an indication about the impact of the proposed scheme on the routing performance, we first evaluate it via an extended version of our generic model for Kademlia routing [RSS15, SRS14b]. After that, we extend the evaluation by performing both extensive simulations for three Kademlia-type systems as well as measurements on standard and modified KAD nodes.

5.1 Related Work

Kademlia-type systems, motivated by their success and high popularity [SS13a, WK13], were the subject of many studies in the last years. In Chapter 3 and Chapter 4, we discussed prior work on Kademlia-type systems, focusing on measurement-based and system characterization studies. We shift our attention here to prior studies that are more relevant to the contribution that we present in this chapter. In particular, we focus on (i) studies that analysed the routing performance and (ii) those that improved it.

Crosby and Wallach [CW07] measured the lookup latency in MDHT and Azureus¹. Most importantly, the authors reported relatively high lookup latency values, and attributed this result to the high ratio of stale routing information. Similarly, Stutzbach and Rejaie [SR06a] analysed the lookup process and measured the lookup latency in KAD. Steiner et al. [SCB10] studied the lookup latency in KAD taking into account the impact of both external factors (*e.g.* RTT of lookup messages) as well as internal lookup parameters such as lookup parallelism. In [RSS15], we developed a comprehensive formal model allowing the derivation of hop count distribution in the entire family of Kademlia-type systems.

In addition, there are studies investigating the possibility to improve the standard routing performance. For instance, Falkner et al. [FPJ⁺07] suggested to adapt the lookup parameters during the runtime according to the number of expected lookup response messages. Their approach reduced the median lookup latency on the cost of increasing the lookup overhead. The study of Steiner et al. [SCB10], in addition to the aforementioned analysis of lookup latency, improved the lookup latency by coupling the lookup with content retrieval. More recently, Jimenez et al. [JOK11] suggested several modifications to MDHT, with which they succeeded to reduce the lookup latency. In addition, other studies succeeded to reduce the lookup costs, measured by the latency or signalling overhead, via caching [EFK13, WXLZ06], geographical proximity [KLKP08, CDHR02], or recursive lookups [Hee10].

In contrast to the aforementioned studies, we do not focus on optimizing the parameters governing the routing table structure or the lookup scheme. Instead, we aim to develop a low-overhead scheme to improve the lookup performance,

¹ Azureus is the P2P overlay network used by the Bittorrent's Vuze clients [vuz].

that can be easily integrated both with existing Kademia implementations as well as with previous improvements. In particular, we propose to improve the routing performance, in form of reduction in the average hop count by adapting the standard neighbour selection scheme. Although our approach differs from earlier improvements, it is orthogonal and compatible with them. It thus can improve their performance gains further.

5.2 Improving Routing Performance

We introduce here our approach for improving the routing performance in Kademia-type systems: Section 5.2.1 explains the idea of our approach. After that, to confirm the necessity for a modified scheme that maximizes the diversity degrees, we measure in Section 5.2.2 the diversity degrees of real buckets and how far are they from the maximal degree. Next, in Section 5.2.3, we explain how our approach can be implemented in existing Kademia-type systems without changing the key design elements.

5.2.1 The Idea

Our approach to improve the routing performance is based on a modified neighbour selection scheme. We describe here the idea of the new scheme and the logic behind it.

In Kademia, at most k contacts in each routing table bucket share a common prefix. This means that these contacts are associated with a specific range of identifiers, *i.e.* represent a certain section of the identifier space. This way, without further restrictions on the selection of neighbours, it is likely that a standard bucket stores multiple contacts with very close identifiers. This is because selection of neighbours in Kademia is not uniform. More precisely, the nodes tend to choose neighbours of neighbours, and those can be close to each other. Such identifiers represent contiguous portions in the respective identifier space section, whereas there are other portions not covered by the stored contacts.

By storing contiguous contacts within the same bucket, the node's view of the corresponding identifier space section becomes narrow. We propose to improve the routing performance by widening this view. That is, the node should try to maximize the diversity of contacts inside each bucket independently, by choosing contacts whose identifier prefixes are maximally diverse. Then, the expected common prefix length of the closest contact to an arbitrary target identifier is maximized, which should lead to a lower hop count.

We use the term diversity degree of a bucket to indicate how diverse are the prefixes of the identifiers stored in it. In particular, the diversity degree is measured by the number of distinct $\lfloor \log_2 k \rfloor$ bits after the bucket's common prefix,

resulting in a maximal degree of $2^{\lfloor \log_2 k \rfloor}$. That is 3 bits in MDHT and KAD, whereas iMDHT considers 7, 6, 5, and 4 bits for buckets at levels 0, 1, 2, and 3, respectively, and 3 bits for the lower levels. Figure 5.1 shows an exemplary MDHT routing table: the diversity degrees of buckets *A* and *B* are 8 (the maximal value in MDHT) and 4, respectively.

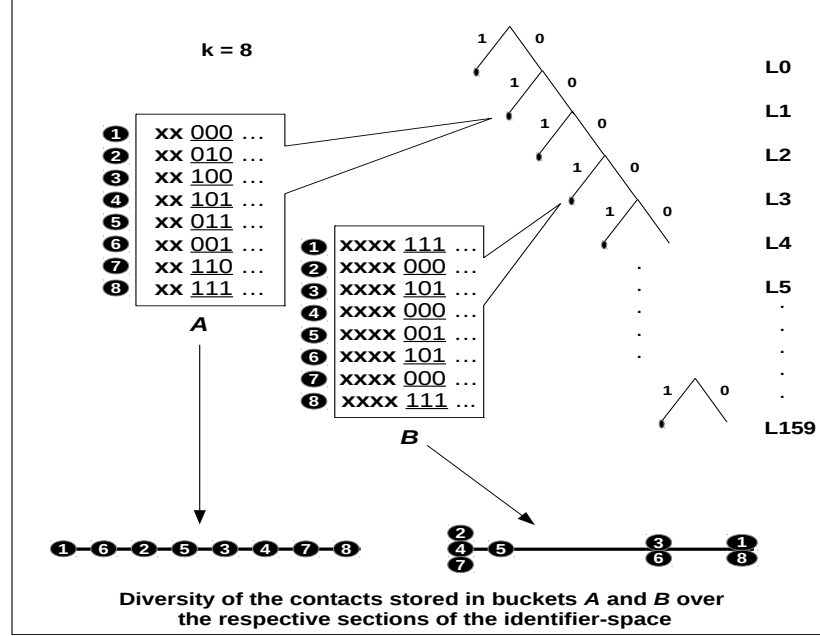


Figure 5.1: Exemplary MDHT routing table: buckets *A* and *B* are located at the second and fourth levels (*i.e.* levels 1 and 3), thus have common prefix lengths of 2 and 4, respectively. Considering the first $\lfloor \log_2 8 \rfloor = 3$ bits after the common prefix, the respective diversity degrees are 8 (the maximal value) and 4.

In [SRS14b] we showed that the expected bit gain (*i.e.* the difference between the node and the closest contact in its routing table to a target identifier x) is increased, thus the average hop count is reduced, by maximizing the aforementioned diversity degree. More precisely, we proved that the expected bit gain offered by the closest contact to x when maximizing the diversity degree is at least as big as the expected bit gain for the standard contact selection. This result is promising and motivates an in-depth evaluation for the new scheme, which we provide in the following sections.²

Note that the idea above is similar to the KAD's routing table structure (Section 2.3), which divides contacts having the same common prefix length with the routing table owner into buckets according to the first 4 bits after the common prefix. However, our approach is more flexible, since it does not restrict the number of prefixes per bucket. Instead, it selects more diverse prefixes, if possible, allowing for a less diverse contact selection if the maximal diversity is not achievable.

² The theoretical analysis of our scheme can be found in [SRS14b].

5.2.2 Diversity Degrees in a Real Kademlia-type System

The aforementioned idea to improve the routing performance could be beneficial only if the diversity degrees of standard Kademlia-type buckets are not already high. We validate in this section our expectation that buckets in Kademlia-type systems are likely to store multiple contacts with contiguous identifiers, *i.e.* they do not uniformly cover all parts of the respective identifier space sections.

We validate this expectation experimentally as follows: we downloaded routing tables of randomly selected online KAD nodes using KadSpider, and analysed the diversity degrees of their buckets. We restricted our analysis to the fourth-level buckets, for two reasons: First, these buckets are expected to be used much more frequently than buckets at lower levels. In particular, given the routing table structure of KAD (Figure 2.1c), fourth-level buckets cover $\frac{11}{16}$ of the identifier space. Hence, on average, $\frac{11}{16}$ of the lookup requests will use contacts from these buckets. Second, there exist with a very high probability contacts in the system that can fill these buckets with all possible prefixes, which enable them to achieve high completeness [SR06a], thus to achieve high diversity degrees. Lower level buckets, in contrast, are likely to achieve lower diversity degrees, due to the fact that there are no contacts with the same prefixes to fill them, as discussed in Section 4.4.2.

The results that we discuss here represent 1.5 million buckets selected at random from a full crawl of KAD conducted on 12.09.2013. We classify these buckets, by the number of contacts they contained, into three groups: (i) 11.31% of the buckets contained eight contacts each, (ii) 18.04% contained nine contacts each, and (iii) 70.65% contained ten contacts each (*i.e.* complete buckets). For each group, we computed the CDF of buckets with a diversity degree $\leq d$.

Figure 5.2 shows the results: 42% to 67% of the buckets have $d \leq 4$ (*i.e.* half the maximal value), and at most 8% have the maximal value of 8. These results confirm our expectation about diversity degrees in real systems. This means that there is a room to increase the diversity, thus to improve the routing performance.

5.2.3 Implementation

The aforementioned idea to improve the routing performance can be implemented in real Kademlia-type systems by only slightly modifying the standard routing table's maintenance processes (Section 2.3.1). More precisely, when a node has to find new contacts (either to insert them to low-populated buckets or to replace stale contacts), it gives preference to those whose identifiers increase the diversity degrees of the corresponding buckets.

By this, our approach does not induce any additional overhead, except for computing the diversity degrees over the bucket's potential identifiers. It also does

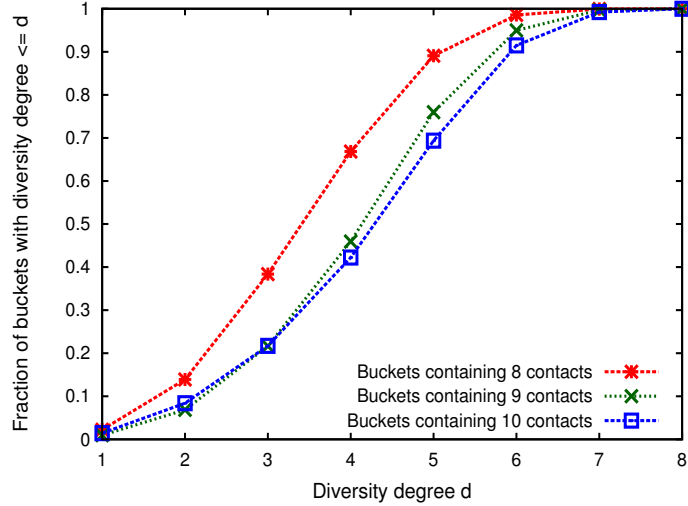


Figure 5.2: CDF of the diversity degrees of standard KAD buckets

not change the original routing table structure nor the original routing protocol. Furthermore, nodes implementing the modified neighbour selection scheme still give preference for long-lived contacts, thus preserving the logic behind this policy (as described in Section 2.1).

5.3 Evaluation

We assess the impact of our modified neighbour selection scheme on the lookup's hop count. The evaluation consists of three parts: First, we evaluate the scheme in a system-wide implementation scenario (*i.e.* all system nodes implement our scheme) in Section 5.3.1. Since none of the existing Kademlia-type systems incorporates our scheme, we perform this part of the evaluation through an extended version of our theoretical model for Kademlia routing [RSS15, SRS14b]. We also validate the model derivations through simulations. Second, we measure the improvement that is gained by our scheme in a real Kademlia-type system (Section 5.3.2). In particular, we measure the routing performance of instrumented KAD nodes implementing our scheme while they are connected to the standard KAD system. Third, we study the impact of system size and churn on the performance gain of our scheme in Section 5.3.3.

5.3.1 Performance Gain: Full Deployment

We evaluate a full deployment of the new neighbour selection scheme, with respect to the lookup's hop count. Such an evaluation is infeasible in real Kademlia-type systems, since none of the lifelike Kademlia-type systems implements our scheme nor we can enforce all system nodes to do so. Therefore, we perform this part of the

evaluation through both theoretical derivations and simulations. Our evaluation considers three exemplary Kademlia-type systems, namely MDHT, iMDHT, and KAD, as they are described and motivated in Chapter 2. We give in the following an overview of the theoretical model, and then we describe our simulation setup, before we discuss the results.

Theoretical Model

The model that we use in our evaluations [RSS15, SRS14b], to the best of our knowledge, is the most comprehensive model for Kademlia routing. It supports various routing table structures and lookup parameters. The main limitation of the model is that it does not support churn. That is to say, it cannot derive the performance gain of our scheme under churn. Still the model derivations can give us an indication about the impact of the scheme on the routing performance.

In short, the model analytically derives the hop count distribution in Kademlia-type systems both for the standard design [RSS15] as well as for the modified one [SRS14b]. More precisely, the model analytically answers the question: *"Given an arbitrary target identifier x and a Kademlia-type system, how likely is it to discover the responsible node within h hops for all h ?"*. That is, the model determines the probability that the path to find the closest node to a target identifier x is of length h .

The lookup process is modelled as a Markov chain: states of the chain represent the common prefix length of the α closest nodes with the target identifier x . Due to the prefix-based routing table structure, the common prefix length is sufficient to determine the transition probabilities. That is, the probability to change from one set of common prefix lengths either to another set of common prefix lengths or to a terminal state. Then, the probability to reach the target in h hops is given by the probability that the Markov chain attains the terminal state after h steps.

Simulation Environment and Setup

We used our simulation model of KAD (Section 4.3.2) as a basis for this part of the evaluation. We extended the original simulation model first by adding MDHT and iMDHT, and secondly by implementing the new neighbour selection scheme. We experimented with 10000 nodes, both without churn and with churn. Following the setup of [BH12], in simulations with churn, session times followed a Weibull distribution with shape $k = 0.5$.

Each online node searched for a random identifier once every 300 simulation seconds. At the beginning of each simulation run, nodes are added to the system until the target size is reached. After an initial stabilization phase of 2000 seconds, the simulator recorded the statistics of interest. All simulation results are averaged over 20 runs.

Performance Results Without Churn: Model Versus Simulations

We first show the results for systems without churn. In particular, we show both the model derivations and simulation results and compare them to each other. Figure 5.3 plots the resulting CDFs of the hop count distributions for the three standard systems as well as for the respective modified ones.

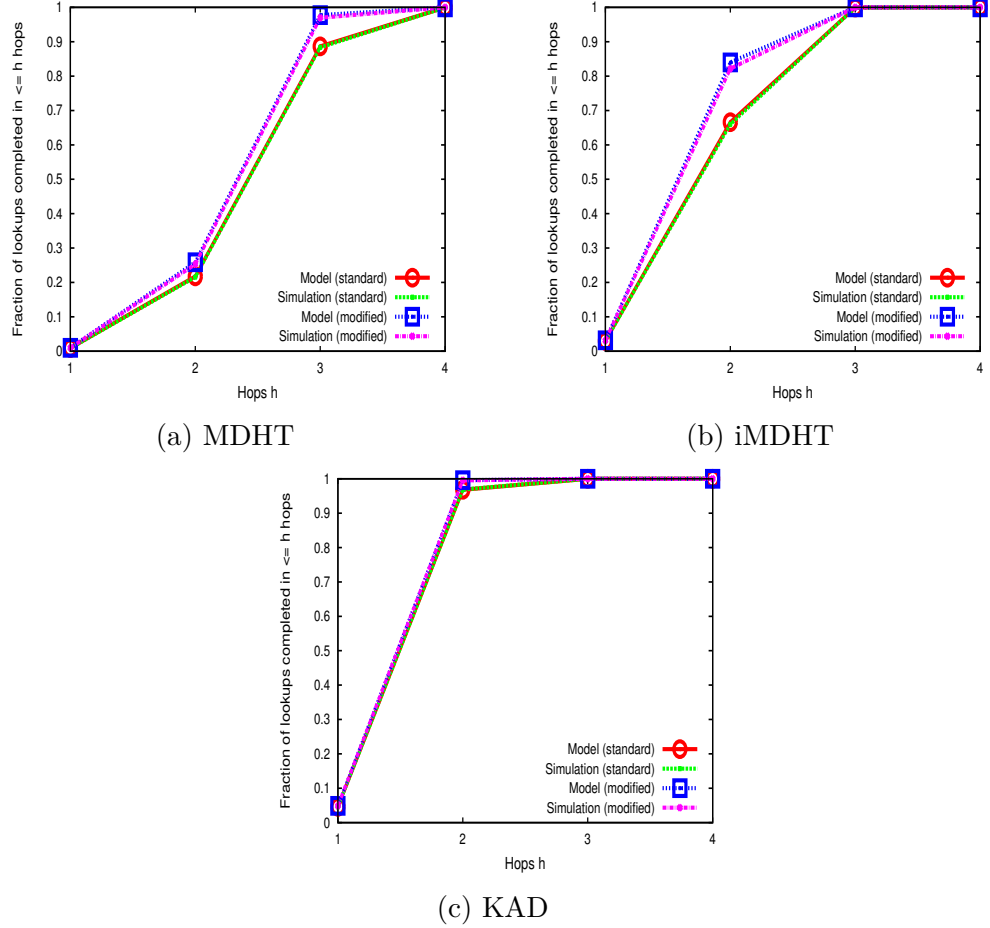


Figure 5.3: CDFs of hop count distributions: model derivations vs. simulations of three exemplary 10000-node Kademlia-type systems without churn

We show in Table 5.1 both the model derivations as well as the corresponding simulation results: (i) the average with the 95% confidence interval (CI), and the median hop count values, and (ii) the hop count improvement achieved by the modified scheme, computed as the difference between "*standard average* - CI" and "*modified average* + CI".

All in all, both the model derivations and simulation results, agreeing with each other, show that the modified systems achieve lower average and median hop count (*i.e.* improve the performance). These results confirm the utility of the proposed scheme.

Table 5.1: Hop count and achieved improvement from model derivations and simulations of three exemplary 10000-node Kademia-type systems without churn

		Simulations		Model
		Average \pm CI	Median	
MDHT	Standard	2.89145 ± 0.00199	3	2.88697
	Modified	2.76611 ± 0.00172	3	2.75416
	Improvement (%)	4.20961	-	4.60025
iMDHT	Standard	2.31044 ± 0.00177	2	2.30470
	Modified	2.14706 ± 0.00146	2	2.12828
	Improvement (%)	6.93672	-	7.65508
KAD	Standard	1.98461 ± 0.00094	2	1.98609
	Modified	1.95545 ± 0.00077	2	1.95535
	Improvement (%)	1.38405	-	1.54760

In addition, it can be seen that the highest performance improvement among the three systems is achieved by iMDHT, whereas the lowest is achieved by KAD. In particular, iMDHT improves the hop count by about 7%, whereas KAD improves it by about 1.5% only, and the improvement achieved by MDHT is in between with about 4.5%. We attribute this disparity among the three systems to the different structures of their routing tables. On the one hand, KAD implements some form of diversity by design (as we discussed in Section 5.2.1), which limits the impact of the additional diversity that is enabled by the new scheme. On the other hand, MDHT and iMDHT do not have such a feature in their standard designs, and therefore they are expected to benefit from the new scheme more than KAD. Comparing iMDHT to MDHT, the buckets in the top four routing table levels in iMDHT (the ones that are expected to be mostly used, Figure 2.1c) can hold more contacts. They hence are expected to have relatively much higher diversity degrees than the respective buckets in MDHT, which explains the higher routing performance improvement achieved by iMDHT.

5.3.2 Performance Gain: Partial Deployment in KAD

We now evaluate the impact of our neighbour selection scheme on the routing performance in a real Kademia-type system. However, as we discussed earlier, measuring the routing performance with our scheme on a system-wide scale is infeasible. Instead, we measure the routing performance achieved by modified nodes implementing the new scheme, during their participation in a standard Kademia-type system. Although the modified nodes are expected to benefit from the new scheme, we expect their gains to be limited in comparison to what could

be achieved with a full deployment. We describe in the following the setup of our measurements, and after that we discuss the results.

Measurement Environment and Setup

We performed our measurements in KAD using four nodes, all of them connected to KAD at the same time. Two nodes used the standard KAD code as implemented in the eMule client software, while the other two nodes implemented our neighbour selection scheme.

During every measurement run, each node issued 512 lookup requests, one every three seconds. The target identifiers are selected such that their hashes are uniformly distributed over the identifier space. In particular, we used 512 keywords from the keyword list of Steiner et al. [SCB10].³ During the measurements, we reported (i) the number of hops traversed by each lookup request and (ii) the diversity degrees of fourth-level routing table buckets. We performed in total 40 measurement runs at several times of the day. That is, the results below are sampled over $512 \times 40 = 20480$ lookups per node. The average population of KAD during the measurements (as measured by KadSpider) was 371,811 active nodes.

General Performance Results

The results that we obtained from the standard nodes are very close to each other, and the same holds for the results of the modified nodes. Therefore, we show and discuss below the results of one standard node and one modified node only.

Figure 5.4 plots the CDFs of the hop count both for the standard node and for the modified one. The results obviously show the improvement achieved by the modified node: more lookups complete in fewer hops, in comparison to the standard scheme. For instance, about 68% of the lookups using the modified scheme completed in three hops or less, in comparison to about 53% using the standard scheme.

We summarize in Table 5.2 the related statistics: (i) the average with 95% CI and median hop count, (ii) the improvement achieved by the modified node over the standard node, computed as the difference between "*standard average* - *CI*" and "*modified average* + *CI*", and (iii) the average diversity degrees of the fourth-level buckets.

We also can see that the gained improvement correlates with the diversity degree: the average diversity degree in the modified node was 7.37, which is very close to the maximal value of 8 in KAD. The corresponding value in the standard node was 4.68 (close to the results that we discussed in Section 5.2.2).

³ The used keywords are listed in Appendix A.

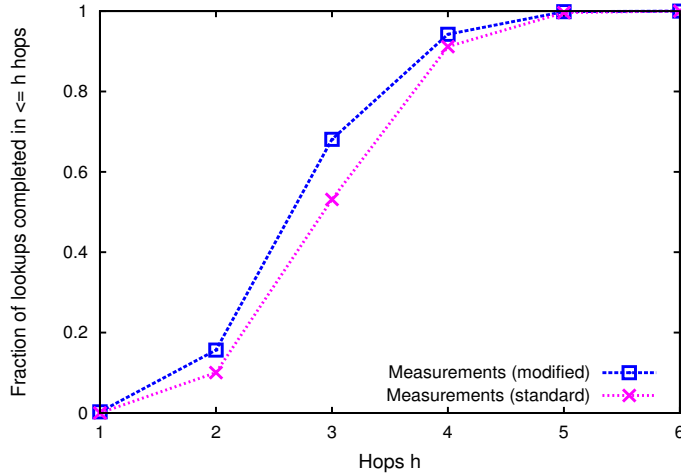


Figure 5.4: CDFs of hop count distributions: measurements on standard and modified KAD nodes

Table 5.2: Hop counts, diversity degrees, and achieved improvement: measurements on KAD nodes

	Standard node	Modified node
Median hop count	3	3
Average hop count	3.39862 ± 0.01583	3.23172 ± 0.01310
Minimum improvement (%)	-	3.36571
Maximum improvement (%)	-	7.68003
Average improvement (%)	-	4.07861
Average diversity	4.68	7.37

5.3.3 Impact of Churn and System Size

In general, one would expect the performance improvement achieved by the modified neighbour selection scheme to increase when it is implemented in a larger network. However, the modified KAD nodes in the measurements, although connected to a standard KAD system, achieved higher improvement than the modified KAD nodes in the full deployment scenario. This result can be attributed to the disparity between the simulation and measurement setups with respect to (i) system sizes and/or (ii) churn rates. In particular, the simulation results above represent only 10000 nodes without churn, while the measurements were conducted in the real (dynamic) KAD system with an average population of 371,811 active nodes. We expect the hop count improvement of our scheme to increase with larger system sizes and/or with higher churn rates. This is because the chance becomes higher to encounter more contacts whose identifiers increase the diversity degrees of routing table buckets.

Impact of Churn

To analyse the impact of churn, we simulated the three systems with a fixed system size of 10000 nodes, and varied the churn rate in each experiment. In particular, we experimented with five different session's average life times: 100000, 50000, 20000, 10000, and 5000 seconds. We plot the average hop count reductions achieved by the three modified systems in Figure 5.5. We also give more detailed results for two exemplary life time settings in Table 5.3.

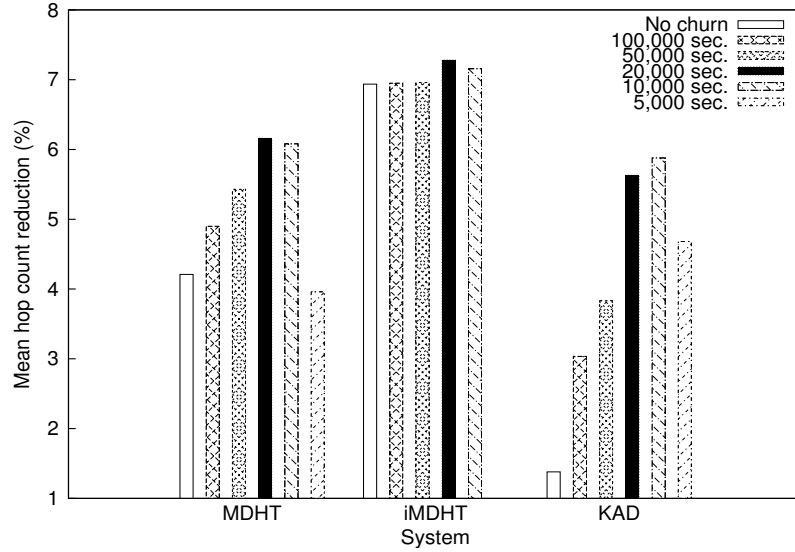


Figure 5.5: Simulative average hop count reduction achieved by three 10000-node modified Kademlia-type systems under churn with different average life times

The comparison results of the different churn settings, both to each other as well as to a system without churn (Table 5.1), show that the average and median hop count without churn are always lower. The results also show that systems with lower churn rates (*i.e.* with longer average life times) outperform those with higher churn rates. These results agree with prior results (*e.g.* [SR06b, BH12]).

With regard to the churn's impact on the hop count improvement, it can be seen in Figure 5.5 that the improvement increases with higher churn rates till some point, and it then decreases for highly aggressive churn rates. This result confirms our hypothesis about the positive impact of churn rate on the achieved hop count improvement. We disregard the reduction in improvement that is observed under very short average life times, *i.e.* under aggressive churn rates. This is because the corresponding results represent only a small fraction of lookup requests, *i.e.* reported while the success rate of lookup requests was very low.

We also can observe in these results that the correlation between the churn rate and the achieved improvement is very small in iMDHT. The reason for this can probably be found in the extremely large bucket size on high routing table

Table 5.3: Hop count and achieved improvement from simulations of three Kademlia-type systems of size 10000 for two exemplary average life times

		Average life time	
		100,000 sec.	20,000 sec.
MDHT	Standard: median	3	3
	Modified: median	3	3
	Standard: average \pm CI	2.97939 ± 0.00219	3.21440 ± 0.00288
	Modified: average \pm CI	2.82948 ± 0.00185	3.01128 ± 0.00234
	Improvement (%)	4.89921	6.16216
iMDHT	Standard: median	2	3
	Modified: median	2	2
	Standard: average \pm CI	2.38366 ± 0.00192	2.57993 ± 0.00245
	Modified: average \pm CI	2.21453 ± 0.00163	2.38774 ± 0.00200
	Improvement (%)	6.95244	7.28376
KAD	Standard: median	2	2
	Modified: median	2	2
	Standard: average \pm CI	2.05295 ± 0.00127	2.22060 ± 0.01910
	Modified: average \pm CI	1.98838 ± 0.00101	2.09217 ± 0.00160
	Improvement (%)	3.03599	5.63017

levels (the most used ones). It takes very long to fill these buckets, and hence it is hard to keep the contacts alive under churn, so that the high stale entry rate impairs further improvement.

Impact of System Size

We now analyse the impact of the system size on the performance gain of our scheme by simulating both the standard and modified systems. We experimented with the three aforementioned Kademlia-type systems without churn, with five different system sizes: 1000, 5000, 10000, 15000, and 20000 nodes. The calculated hop count improvements, however, did not show a correlation between the system size and the performance gain of the new scheme. We confirmed this result through model derivations of different system sizes, up to 100000 nodes. We show in Table 5.4 the model derivations for four exemplary network sizes: 10000, 15000, 20000, and 100000.

Table 5.4: Model derivations for different system sizes: hop count and improvement

	Size	Standard	Modified	Improvement (%)
MDHT	10000	2.88697	2.75416	4.60025
	15000	2.99908	2.84116	5.26561
	20000	3.07960	2.91306	5.71701
	100000	3.53212	3.36691	4.90687
iMDHT	10000	2.30470	2.12828	7.65508
	15000	2.43094	2.25572	7.76781
	20000	2.51711	2.36458	6.45062
	100000	2.94909	2.80668	5.07397
KAD	10000	1.98609	1.95535	1.54760
	15000	2.04304	1.97799	3.28869
	20000	2.09596	2.00861	4.34878
	100000	2.49518	2.41235	3.43358

5.4 Summary

We analysed in this chapter a previously disregarded routing performance factor in Kademlia: the diversity of neighbours over the corresponding identifier space sections. We showed that the average diversity in a real Kademlia-type system is almost half the maximal value, and also that maximizing this diversity can improve the routing performance. Consequently, we proposed a new neighbour selection scheme in which each node attempts to maximize the diversity within each routing table bucket independently. Theoretical results, simulations, and measurements of modified KAD nodes confirmed the utility of our scheme in form of reduction in the average hop count.

Part II

Improving Named-Data Networking

Overview of Named-Data Networking

Information Centric Networking (ICN) is a promising research direction that aims to move the Internet towards a content distribution architecture in order to better match the current Internet usage. In particular, ICN proposes for a receiver-driven content-centric communication model that substitutes the current sender-driven host-centric model.

Several ICN-based architectures emerged in recent years [KCC⁺07, JST⁺09, LVT10, Dan09]. The proposed designs have two ideas in common: (i) in-network caching and (ii) accessing content by name. Named-Data Networking (NDN), in particular, is widely looked at as a potential architecture for the future Internet. This is mainly because its design addresses several technical issues and provides answers for many relevant questions.

We exclusively deal with NDN in this part of the dissertation, with the goal to improve its performance and robustness. In the remainder of this chapter, we give an overview of NDN, and then we describe its key design concepts and features.

6.1 Overview of NDN

Named-Data Networking (NDN) is one of five projects funded by the American's National Science Foundation (NSF) for Future Internet Architectures (FIA) [fia]. It is based on the Content-Centric Networking (CCN) architecture [JST⁺09] which was initiated at Xerox PARC by Van Jacobson and others, with the goal to coincide the host-centric design of the Internet with its current content-centric usage.¹

As can be seen in Figure 6.1, NDN makes named contents the thin-waist of its protocol stack. The stack also includes two new layers: (i) the strategy layer and (ii) the security layer. The strategy layer is responsible for forwarding and transporting functions. It is located above the underlying networking technologies including the

¹ The terms NDN and CCN are commonly used interchangeably in the literature.

IP itself. This way, NDN enables to adapt forwarding and transporting services according to the access network and applications. It also enables to optimize link selection in multi-homed nodes under dynamic connectivity conditions [JST⁺09]. As for the security layer, it implements security functions on the content itself rather than on the communication channels.

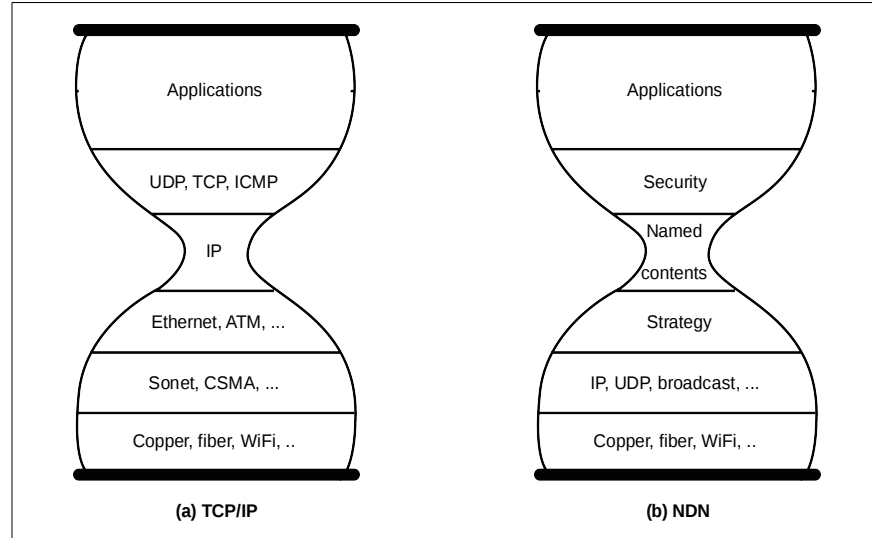


Figure 6.1: Protocol stacks of: (a) TCP/IP and (b) NDN (adapted from [JST⁺09])

6.2 Operation Primitives and Packet Types

The communication model in NDN, similar to other ICN architectures, has its roots in the publish/subscribe (pub/sub) paradigm [EFGK03] which decouples content requests and responses with respect to both location and time. Such decoupling enables for location-independent and asynchronous content delivery.

NDN relies on two operation primitives: register and interest, mimicking publish and subscribe, respectively. More precisely, NDN implements a consumer-driven content-centric communication model, using two packet types: interest packets and data packets (Figure 6.2). The consumer uses an interest packet to request a named content by its name. The content itself is then delivered inside a data packet on the same path through which it was requested in the reverse way. At most one data packet can be retrieved per interest packet.

6.3 Content Naming

Each content in NDN is identified, requested, and accessed by a unique name, rather than location or host address. NDN uses hierarchical

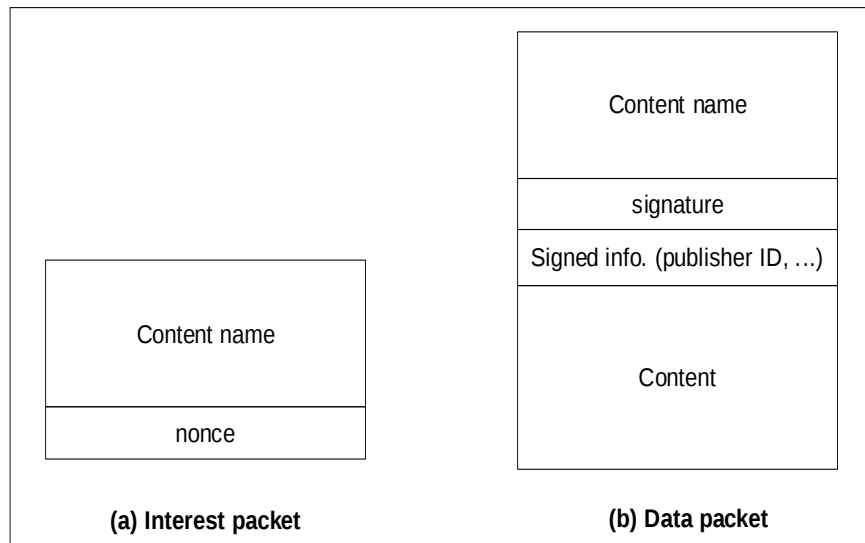


Figure 6.2: NDN packet types: (a) interest packet and (b) data packet (adapted from [JST⁺09]). The nonce field in the interest packet is used together with the content name to identify looping interest packets.

content names, *e.g.* `"/de.tu-darmstadt/dissertations/salah.pdf"`. In this example, `"/de.tu-darmstadt"` is a globally-routable name while `"/dissertations/salah.pdf"` is an organizational name. In contrast to flat names, hierarchical names enable for name aggregation, which reduces the number of routing entries. Hierarchical names also can be human-readable.

6.4 In-network Caching

Similar to other ICN architectures, in-network caching is a key design element in NDN. In particular, NDN implements on-path (or en-route) in-network caching. That is, when a content is retrieved, a copy of it is cached in each node along the path from the content provider to the consumer. Decisions regarding content admission and replacement are taken (and executed) by each node independently, according to its local view of content properties and the cache state (*i.e.* names of contents in the node's cache).

6.5 Node Model

The node model in NDN consists of three data structures:

1. Content Store (CS) or cache: It temporarily holds data packets passing through the node.

2. Pending Interest Table (PIT): It maintains content names of recently received but not yet satisfied interest packets. That is to say, NDN nodes do not only cache contents but also cache the corresponding requests. Each PIT entry also specifies the incoming interface(s) through which the corresponding interest packet was received. A PIT entry is removed either when the corresponding data packet is received or when its timeout is caught.
3. Forwarding Information Base (FIB): Mimicking a routing table, FIB maintains a list of potential outgoing interfaces for different content names (or name-prefixes).

6.6 Packet Handling

With the aforementioned node model, interest packets and data packets are handled in NDN as follows. On the one hand, upon receiving an interest packet, the node looks for a matching name in its CS. If it is found, the node forwards the corresponding data packet to the same interface from which the interest packet was received. Otherwise, the node looks for the name in its PIT. If a matching entry is found but the interface from which the interest packet was received is not listed, the new interface is appended to the same entry, and nothing otherwise. This way, the nodes avoid forwarding duplicate copies of identical interest packets. If no matching PIT entry is found, a new one is created, then the FIB is consulted, and lastly the packet is routed accordingly.

On the other hand, when receiving a data packet, the node first looks for the content name in its PIT. If it is found, the data packet is cached in the CS, then the packet is forwarded to the listed interfaces, and lastly the respective PIT entry is deleted. If no matching PIT entry is found, the packet is discarded.²

Figure 6.3 provides an example illustrating the node model and how packets are handled in NDN.

6.7 Content-based Security

Since each content is identified and accessed in NDN by its unique name regardless of its location, NDN applies content-based security mechanisms. That is, NDN implements security on the content itself instead of securing the communication channels. More precisely, with the aforementioned security layer, content authenticity and integrity are dealt in NDN via a digital signature added to

² Note that with the above described routing and forwarding procedures there is no need to include addresses in NDN packets.

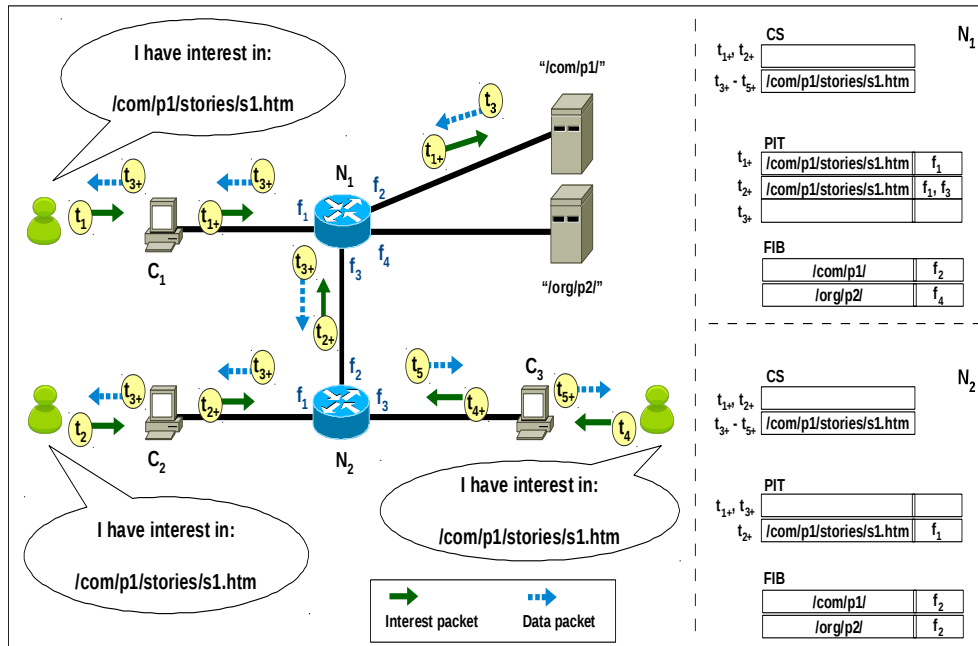


Figure 6.3: Handling interest and data packets in NDN – an illustrative example. Contents of CS and PIT after each time-stamp and FIB of N_1 and N_2 are shown. t_{i+} denotes the time after a time-stamp t_i till before t_{i+1} . The interest packet sent by C_2 is not forwarded further by N_1 since a matching PIT entry is found. The interest packet sent by C_3 is consumed by the matching cached copy that is found in N_2 .

each data packet (Figure 6.2).³ The signature is computed by the origin content provider over the content name and the content itself, thus binding them with each other. The creator's public key can be retrieved using the information contained in the packet (Figure 6.2). This way, the packet's authenticity and integrity can be verified regardless from where the packet is retrieved.

The above described properties of NDN, namely: in-network caching, the stateful forwarding plane (using PITs), and name-based routing, make the network robust to several types of traditional DDoS attacks. For instance, reflection attacks, bandwidth depletion, black-holing, and prefix hijacking are eliminated or at least mitigated in NDN, by design (see [cis14] for explanation). Furthermore, since NDN does not rely on a name resolution service, DNS cache poisoning does not represent a threat in NDN.

6.8 Summary

We gave in this chapter an overview of the key design elements and features of NDN, a potential architecture for the future Internet. NDN, same as other ICN

³ Interest packets, in contrast, do not include a signature field. Hence, their origin and integrity are unverifiable.

architectures, aims at matching the host-centric design of the Internet with its current content-centric usage. Towards this end, NDN implements four ideas: networking named content, on-path caching, consumer-driven communication model, and content-based security.

In the next two chapters, we modify the original design of NDN aiming at (i) realizing coordinated caching and cache-aware routing (Chapter 7) and (ii) making the system robust in face of a new type of DDoS attack called the interest flooding attack (Chapter 8).

Coordinated Cache Management in NDN

In-network caching is a common design element in ICN architectures with which they aim to improve the efficiency gains of content distribution over the Internet. Proposed in-network caching schemes perform cache management autonomously (*i.e.* they are uncoordinated or uncooperative). That is, each caching node (or node) independently caches part of the contents passing through it according to (i) its local view of content access information (*i.e.* names of requested contents and their properties such as frequency or recency of requests) and (ii) names of locally cached contents (hereafter, local cache state). For instance, in NDN [JST⁺09] contents are cached by all nodes on the path from the content provider to the consumer. This scheme is commonly known as on-path caching.

Autonomous cache management, although simple and does not cause coordination overhead, has three drawbacks: First, it is likely to cause unnecessary large cache redundancy; it thus poorly utilizes available (already limited [PV11]) cache space. Second, it is very likely to result in suboptimal selection of contents that are given priority for caching, as the node's view for content access is likely different from the network-wide one. Third, it implies cache-ignorant routing. That is, the system can take advantage of cached contents only when they are located on the default routes of interest packets which requested them.

These drawbacks already raised doubts about achieving the intended gains of in-network caching in ICN. In particular, prior results (e.g [RR11, SLYJ13]) already showed that in some cases only a very small fraction of content requests hit caches. Consequently, several studies (*e.g.* [SLYJ13, WLV⁺12, SFT13, BKST13]) proposed to address the aforementioned drawbacks through cache coordination.

We argue that coordinated cache management can be beneficial only when it is based on timely and global (*i.e.* network-wide) knowledge of (i) content access information and (ii) cache states (*i.e.* which content is cached by which node?). In particular, timely and global knowledge of content access information and cache states enables to realize network-wide caching goals and cache-aware routing.

However, the distributed nature of the Internet beside the massive volumes and high dynamics of coordination information render such coordination (hereafter, global coordination) impractical. Therefore, despite the considerable amount of prior attempts to coordinate caching in ICN (see [PCP12, LSS04, CHPP12, SLYJ13, LS11, LSS04, ENM⁺12, LCS06, SFT13, BKST13] and the references therein), they either favour low coordination overhead over high efficiency gain or vice versa. That is to say, a solution that realizes a global coordination with an affordable coordination overhead is still missing, to the best of our knowledge. Realizing such a solution is the goal that we aim to achieve here.

Towards this end, we present and evaluate CoMon, a framework for **C**oordination that is based on (i) lightweight **M**onitoring of content access information, (ii) bounded advertisement of cache states, and (iii) centralized cache coordination.

In essence, CoMon is designed to work inside an Autonomous System (AS), and it assigns traffic monitoring and re-routing tasks to a small number of nodes only. Monitoring nodes are selected such that the entire traffic passes (or is enforced to pass) through them at an early stage. Selected monitoring nodes work with a controller that takes AS-wide decisions in a centralized way.

7.1 Motivating Example

This example illustrates the benefits and challenges of coordinating caching decisions in NDN¹. The example compares the performance of four different systems: a system without coordination, a system applying global coordination, and two systems applying partial coordination. In both the two systems with partial coordination, all nodes participate in coordination. They differ from each other in the coordination-related information that nodes exchange among each other. In the first (hereafter, partial 1), the nodes exchange only content access information, while in the second (hereafter, partial 2) the nodes exchange only cache states.

On the one hand, both the uncoordinated system and partial 1 do not support cache-aware routing. Instead, interest packets are forwarded along the shortest path towards the origin content provider. This way, interest packets are consumed either by the provider or by a cache on the path. Each node in those systems ranks the observed contents, according to its knowledge of content access information and cache states, and then caches the top ranked content. On the other hand, the coordinated system and partial 2 implement cache-aware routing. That is, interest packets are forwarded towards the other node if the content is cached there or

¹ This example extends on an example from [LXWZ13]. Our example includes more cases and evaluation metrics.

towards the content provider otherwise. The two systems aim at maximizing the diversity of cached contents. In practice, each node ranks the observed contents according to its knowledge of content access information and global cache states, and it caches the top ranked content unless it is cached by the other node. In the latter case, the node caches the second ranked content.

In the example, as shown in Figure 7.1, there is an AS called A containing two consumer nodes N_0 and N_1 (through which clients access the network and contents), in addition to a gateway node N_2 (connecting A to other networks including the Internet). N_0 and N_1 has a cache store (CS) that is capable of holding one content each, whereas N_2 has no CS. N_0 and N_1 use LFU (least frequently used) for content replacements in caches; in case of equality, they use LRU (least recently used). C_0 and C_1 abstract clients connecting to N_0 and N_1 , respectively. The clients issue seven interest packets in total at the specified time-stamps. The server S abstracts all origin content providers. Figure 7.1 also shows at each t_{i+} ² the content cached by N_0 and N_1 .

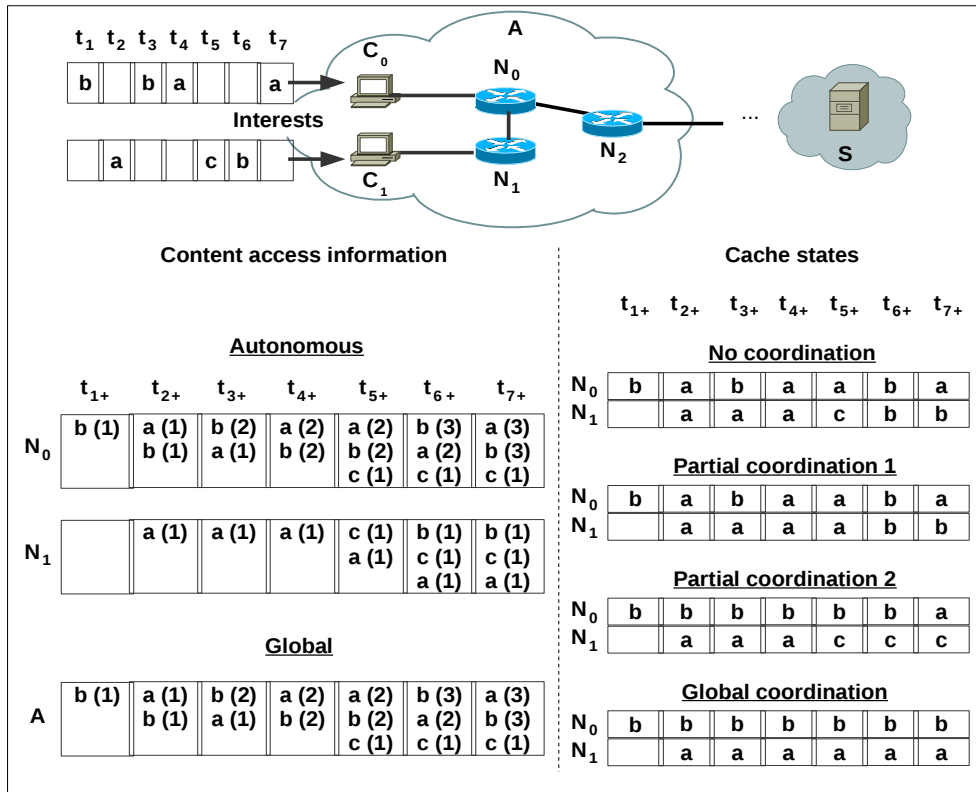


Figure 7.1: Coordinated caching – a motivating example

² t_i denotes the time-stamp at which each interest packet was issued, while t_{i+} denotes the time between a time-stamp t_i and the next time-stamp t_{i+1} .

We compare the four systems by using four metrics widely used in the literature:

1. Cache diversity: The ratio of unique cached contents to the total number of cached contents. By maximizing cache diversity, the network operator aims at reducing the number of contents that are retrieved from outside the domain. However, this is not always the primary goal. For instance, some systems allow for cache redundancy with the goal to make cached contents closer to customers (Wang et al. [WBK14] discuss the trade-off).
2. Server hit ratio: The ratio of interest packets consumed by origin content providers to the total number of interest packets. Minimizing the server hit ratio leads to minimizing the network traffic that crosses other ASs. This is important since the transit cost is commonly charged according to the traffic volume [DPM11].
3. Normalized hop count: The number of hops traversed by a data packet normalized over the number of hops between the consumer and the origin content provider. Since several latencies are incurred at each hop (*e.g.* cache lookup, PIT lookup, routing table lookup, forward), network providers attempt to minimize the hop count (unless it contradicts some other goals).
4. Content replacements: The total number of content replacements in caching nodes. Fewer content replacements are preferred since it means fewer computations and less access to the memory.

Table 7.1 summarizes the results, comparing the four systems after all caches became full (from t_3 onwards). We assume that the distance between N_2 and S is 10 hops. The results show that the system with full coordination outperforms the other systems (for all four metrics), and that the two systems with partial coordination (remarkably partial 2) outperform the system without coordination.³ However, this superiority comes with a price (*i.e.* the coordination overhead) represented by the number of messages that need to be exchanged (between N_0 and N_1) and processed to coordinate caching and routing decisions.

7.2 Related Work

We discuss in this section prior work on cache coordination in ICN. For a comprehensive overview and taxonomy of in-network caching schemes, the

³ Note that N_0 's local view of content access without coordination is the same as with global coordination. This is attributed to N_0 's central position in the network, and because none of the interest packets issued by C_1 were consumed by N_1 's cache. Nevertheless, as can be seen in the results, this knowledge alone was not sufficient for N_0 to take decisions as efficient as if the cache state of N_1 was also known.

Table 7.1: Coordinated caching: performance results of the motivating example

Cache diversity	
No coordination	$(1+0.5+1+0.5+1)/5 = 80\%$
Partial coordination 1	$(1+0.5+0.5+0.5+1)/5 = 70\%$
Partial coordination 2	$(1+1+1+1+1)/5 = 100\%$
Global coordination	$(1+1+1+1+1)/5 = 100\%$
Server hit ratio	
No coordination	$(1+1+1+1+1)/5 = 100\%$
Partial coordination 1	$(1+1+1+1+1)/5 = 100\%$
Partial coordination 2	$(0+0+1+0+1)/5 = 40\%$
Global coordination	$(0+0+1+0+0)/5 = 20\%$
Normalized hop count	
No coordination	$(12+12+13+13+12)/62 = 100\%$ ⁴
Partial coordination 1	$(12+12+13+13+12)/62 = 100\%$
Partial coordination 2	$(1+2+13+2+12)/62 \approx 48\%$
Global coordination	$(1+2+13+2+2)/62 \approx 32\%$
Replacements per node	
No coordination	$(1+1+0+1+1+1+1)/7 \approx 86\%$
Partial coordination 1	$(1+1+0+0+1+1+1)/7 \approx 71\%$
Partial coordination 2	$(0+0+0+1+0+0+1)/7 \approx 29\%$
Global coordination	$(0+0+0+0+0+0+0)/7 = 0$

reader is referred to [ZLL13] and [XVS⁺13]. Existing solutions for coordinating caching-related decisions can be classified as either indirect (implicit) or direct (explicit). Both classes aim to outperform uncoordinated caching.

In general, indirect coordination schemes favour low coordination overhead over high efficiency gains. In these schemes, nodes do not exchange information among each other or exchange little information. Instead, coordination is realized implicitly. A notable scheme for indirect cache coordination is probabilistic caching. In this scheme, returned contents are cached at each node on the delivery path either with a fixed probability [LSS04] or with a probability determined by some metric such as the node's distance from the consumer (the closer the node the higher the caching probability) [PCP12]. Another indirect scheme is centrality-based caching in which contents are cached by the most central node on the delivery path [CHPP12]. Centrality here is calculated by some metric

⁴ The denominator (number of hops till S) = $12+12+13+13+12 = 62$

such as betweenness centrality⁵. Alternatively, hash-based solutions use a hash function to map content names with node identifiers [SLYJ13, LS11]. Interest packets are then routed to potential cached copies using the same function. The simplest indirect scheme is to cache returned content in a node locating on the path selected randomly [ENM⁺12]. There are other indirect schemes activated after a cache hit. For instance, after a cache hit, the cached content is moved to the direct downstream node (Move Copy Down (MCD) [LSS04]), or another copy is cached at the direct downstream node (Leave Copy Down (LCD) [LCS06]).

In direct coordination schemes, in contrast, nodes make caching-related decisions based on the information which they exchange explicitly among each other. Proposed direct coordination schemes work at one of three levels: (i) among neighbouring nodes [WZB13], (ii) among consumer nodes [BKST13, NST13], or (iii) among all nodes [SFT13]. Coordination in the last two cases is usually performed via a (logically) centralized controller. To the best of our knowledge, among the proposed coordination schemes, only the designs of [SFT13], [BKST13], and [NST13] can realize nearly global coordination. To do so, either all network nodes [SFT13] or all consumer nodes [BKST13, NST13] should participate in coordination. The high coordination overhead of those solutions, however, likely makes them impractical. Therefore, [SFT13] and [BKST13] also propose lighter versions of their original (costly) schemes. The lighter versions, however, achieve lower caching efficiency as shown in the same papers.

To conclude, evaluations of indirect coordination solutions showed that they outperform the uncoordinated NDN with low coordination overhead. However, their efficiency gains are still limited in comparison to what can be achieved with global coordination. This is obvious, for instance, when comparing the caching efficiency of the original solutions of [SFT13] and [BKST13] to the respective lighter versions. That said, as yet there is no solution that can realize nearly global coordination with low overhead. This is the gap that we aim to fill.

In addition to the studies above, there has been a little effort to model an optimal cooperation policy for in-network caching in ICN [LXWZ13]. However, a practical solution for realizing such a policy is still missing.

7.3 Design Requirements and Overview

Our goal is to design a cache coordination solution that is efficient and incurs low overhead. To this end, we present a coordination framework called CoMon. CoMon is consistent with NDN. However, the design concepts are applicable for other ICN architectures.

⁵ The BC score of a node v counts the total fraction of all shortest paths in the network that pass through v [Fre77].

In this section, we first list the design requirements of CoMon (Section 7.3.1). After that, we describe the main idea of CoMon as well as its architecture and operation primitives (Section 7.3.2).

7.3.1 Design Requirements

We guide the design of CoMon by the following three requirements:

- R1) CoMon should enable the network operator to realize network-wide caching goals (*e.g.* caching the most popular contents without duplications).
- R2) CoMon should enable for cache-aware routing.
- R3) CoMon should induce a low overhead.

The first two requirements aim to improve the overall caching efficiency gains (i) by increasing the quality and accuracy of caching decisions (R1) and (ii) by increasing the utilization of cached contents even if they are not located on the default routes of interest packets (R2).

While the intended benefits behind R1 and R2 look appealing, fulfilling them is challenging, and also they are contradicting R3. In particular, R1 and R2 imply that nodes should have up-to-date global knowledge of cache states. R1 also implies that nodes should know global content access information. The volume of such information in large networks such as the Internet is significant and causes high signalling overhead. Such an overhead increases in highly dynamic environments since global cache states and content access information must be exchanged frequently. Otherwise, this information becomes outdated, which results in suboptimal caching decisions.

7.3.2 Architecture and Operation Primitives

We aim to find the right balance between the three aforementioned requirements. Towards this end, we propose to achieve this in CoMon by reducing the number of nodes participating in coordination. In particular, CoMon implements coordination within AS, *i.e.* a set of nodes (*i.e.* routers) under a single technical administration entity [HB96]. CoMon assigns coordination-related tasks to a set $M \subset V$ of influential nodes, where $|M| \ll |V|$, with the aid of a centralized controller.

Figure 7.2 shows the system architecture of CoMon. It consists of three principal components: (i) an AS Controller (AC), (ii) NDN Nodes (NNs), and (iii) Monitoring Nodes (MNs). We assume in the following homogeneous cache capacities for all nodes. We let c and C denote the cache capacity per node and the overall cache capacity of the AS, respectively (*i.e.* $C = |V| \times c$).

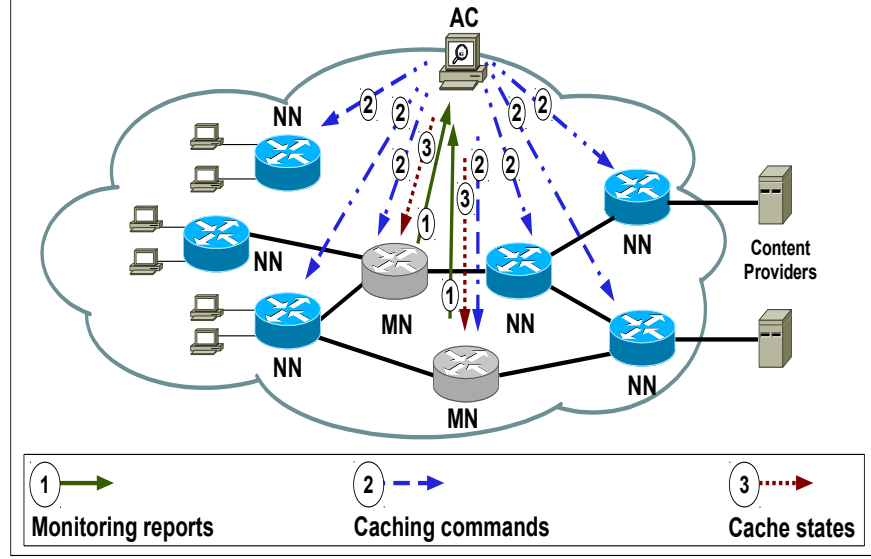


Figure 7.2: System architecture of CoMon: "AC" stands for AS Controller, "NN" stands for NDN Node, and "MN" stands for Monitoring Node

We introduce now these components and describe how they work together to realize AS-wide coordinated caching decisions:

1. AS Controller (AC): Each AS has a controller that (i) aggregates content access information from a set $M \subset V$ of monitoring nodes, (ii) calculates and commands caching-related decisions, and (iii) advertises the cache states in the routing plane of monitoring nodes only.

We implement the AC on a single machine. Instead, the AC can be implemented on multiple machines to distribute the load and to avoid a single-point-of-failure. We leave such a design as a future work.

2. NDN Nodes (NNs): These are similar to NDN nodes [JST⁺09] in that they have routing and caching capabilities. However, instead of taking autonomous caching decisions, each NN caches and evicts contents as commanded by the AC.

In response to a command implying to cache contents, each node (i) identifies which of the contents are not already in its cache, (ii) requests (*i.e.* pre-fetches) each of them using an interest packet and creates a PIT entry initialized without an outgoing interface, and (iii) stores the respective data packets after it verifies their integrity and provenance (to avoid cache poisoning). The NN appends the interface information of matching interest packets that arrive before the NN receives the corresponding data packets to the respective PIT entries.

3. Monitoring Nodes (MNs): A set $M \subset V$ of nodes are selected as MNs. In addition to the routing and caching functions of regular NNs ⁶, each MN (i) monitors interest packets passing through it and records their names and characteristics (*e.g.* number of times a content was requested during the current observation window), (ii) periodically (as often as determined by the AC) uploads the recorded content information (hereafter, monitoring reports) of requested contents to the AC, and clears this information afterwards, and (iii) receives lists of cached contents (*i.e.* global cache states) from the AC, and (re)routes interest packets accordingly.

Note that the system operation described above implies off-path caching (rather than the default on-path caching scheme). By this, CoMon gets rid of NDN's requirements to perform content replacements and signature verifications at line rate.

The control messages which are used for communications between the AC and the nodes (Figure 7.2) are unsolicited data packets. The names used in these packets consist of a reserved name-prefix *"/comon/"*, appended with either *"AC"*, if the packet is destined to the AC, or the identifier of the target node otherwise. Being standard data packets, the AC and the nodes can determine whether control messages are legitimate or not by verifying the contained signatures (Section 6.7).

7.4 Design Specifications

We describe in this section the design specifications of CoMon that are not detailed in Section 7.3.2. In particular, we describe cache-aware routing (Section 7.4.1), caching-related decisions (Section 7.4.2), selection of monitoring nodes and maximizing traffic coverage (Section 7.4.3), and a technique to reduce the coordination overhead (Section 7.4.4).

7.4.1 Cache-aware Routing

CoMon implements Cache-Aware Routing (CAR) as follows:

1. Each interest packet is routed towards the corresponding origin content provider till it encounters an MN.
2. Only that (the first encountered) MN checks whether the requested content is among the currently cached contents or not (according to the most recent global cache states). If not, the original route is preserved.

⁶ We use the term node in a generic way for referring to any node $v \in V$ (*i.e.* either NN or MN), and we use NN for referring to a node u without monitoring nor re-routing capabilities (*i.e.* $u \in V : u \notin M$).

Otherwise, the MN adds the prefix *"/comon/node-name"* to the original content name (where *"node-name"* is the ID of the matching node). For example, if the content *"/de.tu-darmstadt/dissertations/salah.pdf"* is cached at N_7 , the MN changes the name to *"/comon/N7/de.tu-darmstadt/dissertations/salah.pdf"*. Then, the MN re-routes the packet towards that node.

3. In either case, the MN, before forwarding the packet, sets a *"checked"* flag, so that next encountered MNs skip the aforementioned checking step.

CAR requires that each node stores $|V| - 1$ additional FIB entries: one entry towards each of the other nodes in the AS network (so that all nodes can route packets towards each other).

7.4.2 Caching-related Decisions

The AC is responsible for all caching decisions. More precisely, at the end of each observation window (*i.e.* after receiving new monitoring reports from the MNs), the AC calculates caching-related decisions according to the network-wide goals specified by the network operator. In particular, the AC has to determine (i) the contents that are given priority for caching and (ii) their locations in the network. These two decisions implicitly determine the redundancy degree for each selected content as well as its caching duration. That is, each cached copy is kept in the cache till a more recent decision, implying that the content has to be evicted, is taken.

CoMon is compatible with any network-wide goals. Without loss of generality, we describe caching decisions using an exemplary network-wide goal: distributing the most popular contents over the available cache space without duplication. With this a goal, the network operator favours data availability over the hop count required for data delivery inside the network. Similar network-wide caching goals were adopted in several prior studies (*e.g.* [SLYJ13, BKST13]). We describe in the following how CoMon implements the two aforementioned caching-related decisions according to our exemplary goal.

Content Selection

The AC selects contents that are given priority for in-network caching by maintaining a list containing the names of both the currently cached contents and the recently monitored ones. The AC calculates the popularity score f_i for each content i and ranks the contents accordingly. Then (in line with the aforementioned caching goal), the top- C ranked contents are selected for caching.

Content-to-Cache Distribution

CoMon distributes contents to the caching nodes following the observations of Wang et al. [WBK14], with the goal to increase the caching efficiency. In particular, the authors observed that with our aforementioned exemplary caching goal, a high caching efficiency can be achieved when most popular contents are placed in the network core (*i.e.* on the nodes that have highest betweenness centrality (BC) scores) while less popular contents are placed at the network edges [WBK14].⁷

Consequently, CoMon places contents in the network as follows: First, the BC score of each node $v \in V$ is calculated. Second, the nodes are sorted by their BC scores in descending order. Third, the selected contents, ordered from most to least popular, are assigned iteratively to the nodes.

7.4.3 Selection of Monitoring Nodes

Guided by the preset design requirements (Section 7.3.1), MNs should be selected such that:

1. They intercept the entire network traffic. This is essential to collect accurate content access information, thus to improve the quality of caching decisions.
2. Number of MNs (*i.e.* $|M|$) should be kept small. This is because with the above described system operation, the signalling overhead of coordination grows linearly with $|M|$, and the load on AC grows at least linearly with $|M|$.

In general, there is a trade-off between the two conditions above. That is to say, the selection algorithm either increases $|M|$ till covering all routes, or it selects a small $|M|$ without guaranteeing full coverage.

We solve this trade-off in CoMon by (i) developing an algorithm that selects a small number of MNs expected to cover the majority of network traffic, and then (ii) enforcing each interest packets to pass through one of the selected MNs. However, in NDN this design does not guarantee a full traffic coverage, because interest packets can be filtered by caches and PITs before they encounter MNs. Therefore, interest packets that are not monitored before should be forwarded towards MNs even after they are consumed (by PITs or caches).

In the following, we first present our algorithm for selecting MNs. After that, we describe how interest packets in CoMon are enforced to pass through MNs (even after they are consumed by PITs or caches).

⁷ This observation assumes that both content popularity and BC scores exhibit power-law characteristics. Otherwise, coupling between content popularity and network topology might disappear.

Selection Algorithm

Enforcing interest packets to pass through MNs likely will result in hop count overhead (unless the resulting route is identical to the original one). To reduce this overhead, we propose to select MNs not only according to their coverage, but also to give preference for nodes that are located closely to clients.

The corresponding monitor selection problem can be formulated as follows: *"Given a network consisting of a set V of nodes, which set $M \subset V$ should be selected as MNs such that they together cover the majority of network traffic while both $|M|$ and hop count from clients to MNs are minimized".*

Such a problem has been shown to be NP-hard [SGKT06]. Therefore, we develop a new selection heuristic, called Placement based on covered Routes and Closeness to Sources (PRCS). The pseudo-code of PRCS is provided in Algorithm 4, and it can be outlined as follows:

1. Using Dijkstra's algorithm [AMO93], identify the set R of shortest cost paths from each consumer node to each content provider, and then identify the set N of non-gateway nodes that are located on the identified routes (lines: 3 – 7).
2. Calculate the weight $W(n)$ for each node $n \in N$ (lines: 12 – 15), summing n 's partial weights on each route $r \in R$. Considering n 's closeness to the beginning of the route (*i.e.* to the consumer node), its partial weight on r is calculated as follows:

$$w_r(n) = \begin{cases} 1 + \frac{h_r(n)}{l(r)}, & n \text{ is located on } r \\ 0, & \text{otherwise} \end{cases}$$

where $l(r)$ and $h_r(n)$ denote the length of r (*i.e.* hop count) and n 's position on r , respectively. $h_r(n)$ takes the value 0 if n is located next to the gateway node, and it is incremented by 1 with each hop towards the source otherwise. This idea is inspired from [HYEN09].

3. The node with the maximum total weight is then selected as an MN (lines: 16 – 20) and added to the set M (line: 22). The selected node is removed from N (line: 23), and the routes on which it is located are removed from R (line: 24).
4. Repeat step 2 and step 3 till the cardinality of M equals a predetermined value p .

Algorithm 4 Placement based on covered Routes and Closeness to Sources (PRCS)

```

1:  $R \leftarrow \emptyset$  ▷ Set of shortest cost paths
2:  $N \leftarrow \emptyset$  ▷ Set of non-gateway nodes
3: for each consumer-provider pair  $x$  do
4:   Using Dijkstra's algorithm [AMO93], find the shortest cost path  $P_x$ 
5:    $R \leftarrow R \cup P_x$ 
6:    $N \leftarrow N \cup \text{non-gateway nodes on } P_x$ 
7: end for
8:  $M \leftarrow \emptyset$  ▷ Set of MNs
9: while  $|M| \leq p$  do
10:    $max \leftarrow 0$ 
11:   for each node  $n$  in  $N$  do
12:      $W(n) \leftarrow 0$ 
13:     for each route  $r$  in  $R$  do
14:        $W(n) \leftarrow W(n) + w_r(n)$ 
15:     end for
16:     if  $max < W(n)$  then
17:        $max \leftarrow W(n)$ 
18:        $m \leftarrow n$ 
19:        $Z \leftarrow \text{routes that } n \text{ is located on}$ 
20:     end if
21:   end for
22:    $M \leftarrow M \cup \{m\}$ 
23:    $N \leftarrow N - \{m\}$ 
24:    $R \leftarrow R - Z$ 
25: end while

```

Monitor-Aware Routing

Monitor-Aware Routing (MAR) modifies the original routing strategy such that each interest packet crosses at least one MN. We propose two versions of MAR (hereafter, MAR-V1 and MAR-V2). Both versions involve two routing steps, and they differ in the first one.

In MAR-V1, the consumer node first routes each interest packet towards the closest MN. To do that, each consumer node should have an additional FIB entry pointing to the closest MN. The node name of the designated MN is added as a prefix to the original content names of interest packets, so that the packets are forwarded towards the designated MN. Next, once the designated MN receives an interest packet, it removes the aforementioned prefix, and then routes the packet either towards the matching local node if the content is cached or towards the respective origin content provider otherwise. It is obvious that unless the default path contains an MN, MAR-V1 results in additional routing hops.

MAR-V2 aims at minimizing the hop count overhead of MAR-V1. Towards this end, the consumer nodes select the target MN (in the first routing step) according

to the requested content name. This is done using the Finding-First-MN Procedure (Algorithm 5), which we adapt from the Routing-with-Content-Filtering (RCF) algorithm [KLS05]. More precisely, the consumer node in Algorithm 5 passes two parameters to the procedure: (i) its own name and (ii) the name of the requested content. The algorithm can be outlined as follows:

1. It first compares the shortest path costs of the matching paths (from the consumer node towards the corresponding gateway node) ⁸ that contain MNs (lines: 4 – 12).

The total path cost (line: 7) sums up two values: the cost from the consumer node to the MN (line: 5) and the cost from the MN to the gateway node (line: 6).

2. It then selects the path with the lowest cost (lines: 8 – 11).
3. Among the MNs that are located on the selected path, the name of the consumer node's closest MN is returned (line: 13). Then, the consumer node updates its FIB accordingly.

Note that MAR-V1 reduces the number of FIB entries in NNs to $|V| - 1$, since each NN needs only to reach the other $|V| - 1$ nodes (one of them is labelled as closest MN). As for the MNs, the number of FIB entries is the same as when MAR is disabled. The same applies for the number of FIB entries in NNs and MNs when MAR-V2 is enabled.

Forward-Till-Be-Monitored

As discussed above, MAR alone cannot guarantee full traffic coverage. This is because interest packets still can be consumed by a cache or can be captured by a PIT before they encounter an MN. To eliminate the effects of these filters, CoMon implements Forward-Till-Be-Monitored (FTBM), which works as follows:

1. When a node receives an interest packet that is not monitored yet (*i.e.* $checked = 0$) and finds a matching data packet in its cache or a matching PIT entry, the node (i) looks for the closest MN in its FIB, say *"/comon/MNx"*, (ii) adds the prefix *"/comon/MNx/served"* to the original name, and then (iii) forwards the packet accordingly (to *"MNx"* in the example).
2. The designated MN (i) extracts the original content name, (ii) updates content access information accordingly, and (iii) drops the packet. Note that no states (*i.e.* PIT entries) are created for served interest packets.

⁸ We assume that each content's origin provider is accessible through one gateway node only. However, extending Algorithm 5 for multiple gateways is straightforward.

Algorithm 5 Finding the best route's first MN

```

1: // Find the best route from a consumer node  $s$  towards content's corresponding
   gateway node  $t$  that crosses one or more MNs. The procedure returns  $m_c$ , the
   closest MN to  $s$ .
2: procedure FINDING-FIRST-MN( $s, t$ )
3:    $min\_sum \leftarrow largest\_spc$  ▷ spc: shortest path cost
4:   for each MN  $m$  in  $M$  do
5:     compute  $spc(s, m)$ 
6:     compute  $spc(m, t)$ 
7:      $sum = spc(s, m) + spc(m, t)$ 
8:     if  $sum < min\_sum$  then
9:        $min\_sum \leftarrow sum$ 
10:       $m_c \leftarrow m$ 
11:    end if
12:  end for
13:  Return  $m_c$ 
14: end procedure

```

The overhead of FTBM is measured by the hops traversed by the interest packet from the moment it was served till it reached an MN. This overhead does not apply for the respective data packet as the interest packet in this case is already consumed.

7.4.4 Reducing the Frequency of Updates

Changing cache states after each observation window (according to the newly calculated caching decisions) results in signalling overhead for exchanging caching assignments from the AC to the nodes and global cache states from the AC to the MNs. It also causes computational and memory access overhead of updating the MNs' FIBs and consequent content replacements in the caches.

In order to mitigate this overhead, CoMon implements a technique with which the AC first calculates the feasibility of changing (FoC) the set H_c of currently cached contents to the set H_n of newly selected contents:

$$FoC = 1 - \frac{\sum_{x \in H_c, x \in H_n} f_x}{\sum_{y \in H_n} f_y}$$

where f_x and f_y denote the newly calculated popularity scores of contents x and y , respectively. The AC then changes the current configurations only if the resulted FoC value is above some threshold.

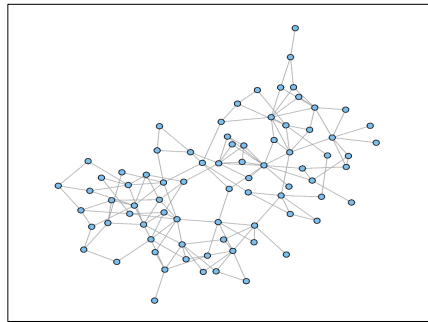
7.5 Evaluation

Our evaluation consists of two parts: First, in Section 7.5.1, we evaluate PRCs (our algorithm for selecting MNs) with respect to both the fraction of routes covered by MNs and closeness of MNs to clients. Second, in Section 7.5.2, we evaluate the caching efficiency gain that can be achieved with CoMon as well as CoMon’s signalling and hop count overhead.

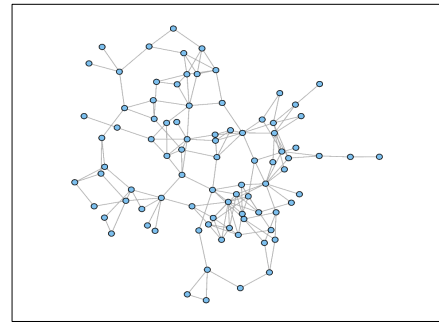
We fed the simulator with real ISP network topologies measured by the Rocketfuel project [SMW02]. These topologies are commonly used by the networking research community. In particular, we experimented with three network topologies: AS-3967, AS-1755, and AS-3257. We summarize their properties in Table 7.2 and plot them in Figure 7.3.

Table 7.2: Basic properties of the three ISP network topologies used in simulations

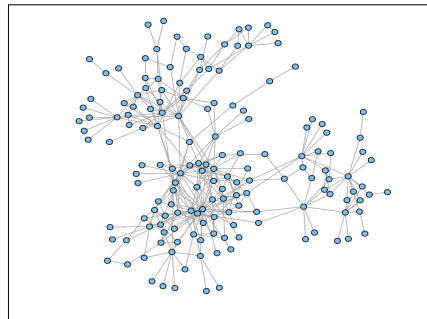
Network	$ V $	$ E $ (bidir.)	Diameter	Avg. path length	Avg. degree
AS-3967	79	147	10	4.08	3.72
AS-1755	87	161	11	4.53	3.7
AS-3257	161	328	10	4.2	4.08



(a) AS-3967



(b) AS-1755



(c) AS-3257

Figure 7.3: The three ISP network topologies used in simulations

We repeated each experiment 20 times. At the beginning of each simulation run, the simulator randomly picks $\lceil 70\% \rceil$ of the nodes as consumer nodes and three of the rest as gateway nodes.

7.5.1 PRCS: Coverage and Closeness to Sources

As we discussed in Section 7.4.3, PRCS is developed with the goal to select a small number of monitoring nodes, close to clients, through which majority of network traffic is expected to pass. We consequently evaluate PRCS with respect to (i) the fraction of routes covered by MNs and (ii) closeness of MNs to consumer nodes.

Figure 7.4 plots the CDF of the fraction of covered routes as a function of the number of MNs (as ranked by PRCS) for an exemplary simulation run. We also compare PRCS with two baseline monitor selection algorithms: (i) betweenness centrality (BC) and (ii) random selection.

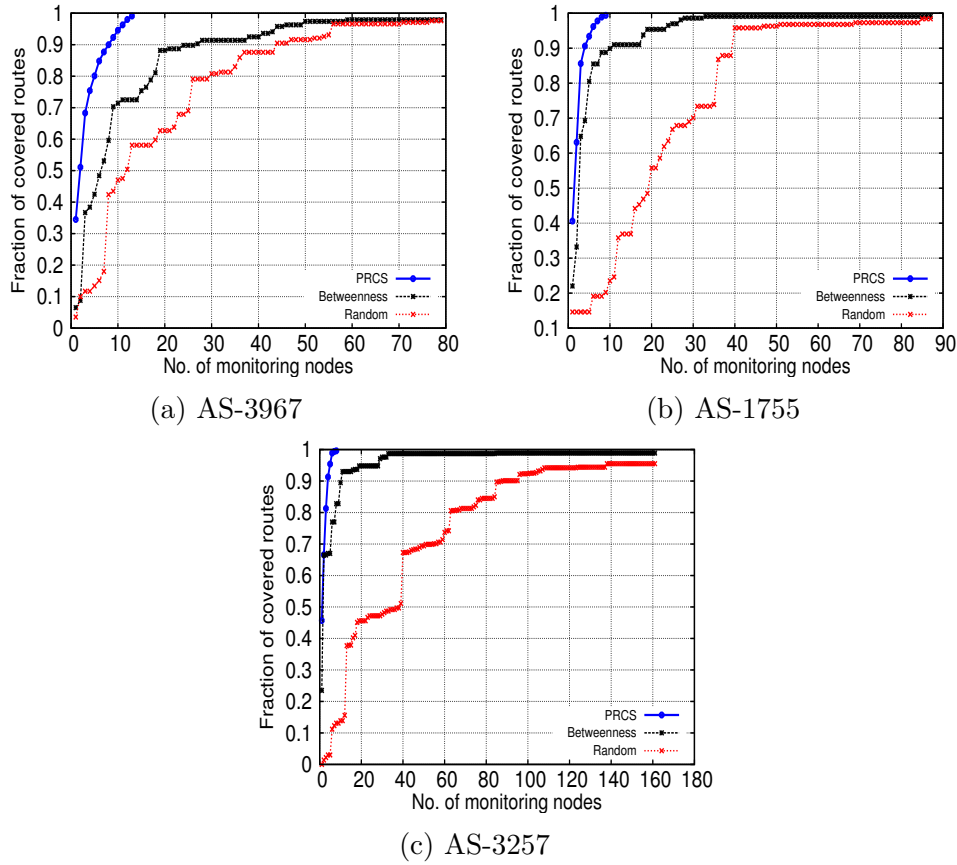


Figure 7.4: PRCS: Fraction of covered routes as a function of the number of monitoring nodes (out of $|V|$)

For instance, in AS-3967 it can be seen that only 13 MNs (*i.e.* less than 17% of the network nodes) are sufficient to cover all routes. That is to say, with those MNs, it is guaranteed that on each route there exists at least one MN. In contrast,

the top 13 BC-ranked nodes and the 13 nodes selected uniformly at random cover only about 73% and 58% of the routes, respectively. The results in the two other network topologies are even better: PRCS in AS-1755 and AS-3257 achieves the same with only 9 MNs and 8 MNs (*i.e.* less than 10% and 5% of the network nodes), respectively. Note that the aggregated knowledge of packets that can be captured by these MNs is equivalent to the aggregate knowledge of all consumer nodes ($\lceil 70\% \rceil$ of the network nodes in our setup).

Table 7.3 summarizes statistics over 20 simulation runs for the percentage of MNs that cover all routes. All results confirm the superiority of PRCS over BC and random selection.

Table 7.3: PRCS statistics: percentage of MNs that cover all routes

	AS-3967	AS-1755	AS-3257
Minimum (%)	13.28	10.01	4.48
Maximum (%)	17.95	14.82	6.83
Median (%)	16.01	12.35	5.33
Average (%)	15.86	12.47	5.53
Std. dev.	1.26	1.34	0.76

We plot in Figure 7.5 the CDF of the distances between the MNs (ranked by PRCS) and consumer nodes. More precisely, for each route, we measure the number of hops after the consumer node (towards the gateway node) till the closest MN. We then normalize this distance by the hop count of the entire route. We can see that the closest MN, for more than 80% of the routes in AS-3967 and more than 70% of the routes in the other two network topologies, are located on the first half of the route, *i.e.* closer to the consumer node than to the gateway. It also can be seen that part of MNs (20% in AS-3967, 14.1% in AS-1755, and 7% in AS-3257) are consumer nodes (*i.e.* distance = 0). Table 7.4 summarizes the statistics over 20 simulation runs of the fraction of closest MNs that are located on the first half of the route.

Table 7.4: PRCS statistics: fraction of the closest MNs located on the first half of the route

	AS-3967	AS-1755	AS-3257
Minimum (%)	77.85	69.54	66.09
Maximum (%)	84.25	78.70	75.17
Median (%)	81.93	74.80	71.97
Average (%)	81.42	74.29	71.64
Std. dev.	2.08	2.99	2.14

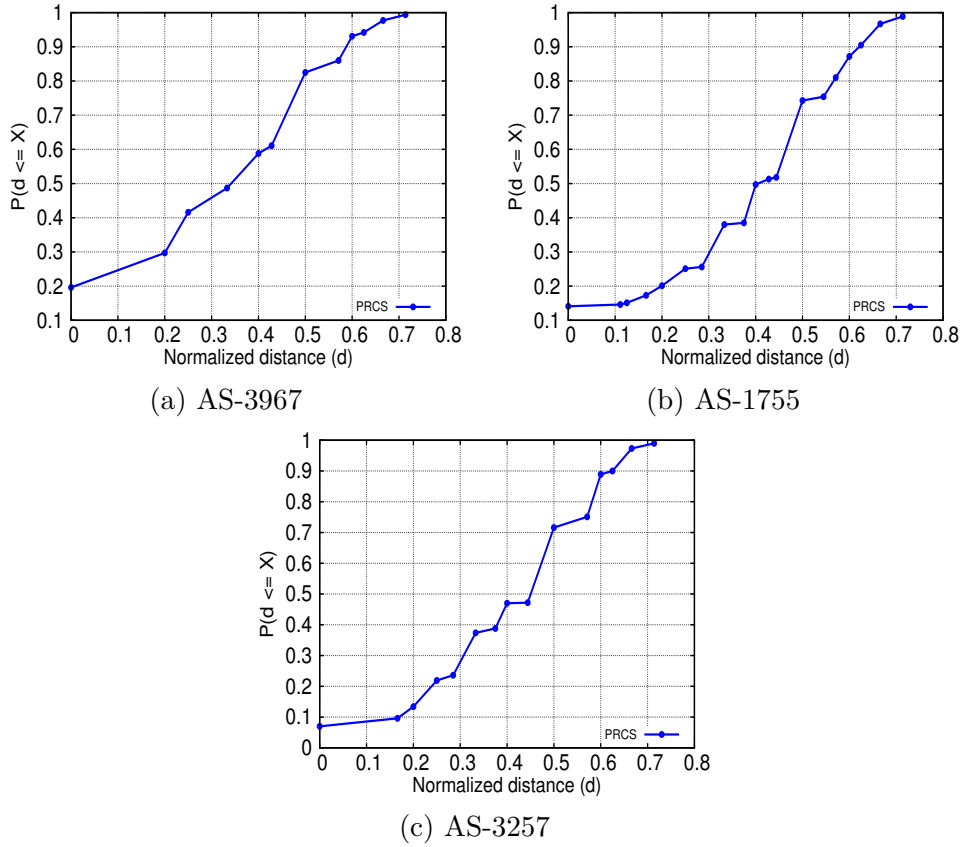


Figure 7.5: PRCS: CDF of hops between consumer nodes and closest monitoring node (normalised by route length)

Based on the results above, it is sensible to conclude that PRCS, to a high extent, achieves the goals for which it was developed (Section 7.4.3). That is, it enables to select few nodes that jointly cover the network traffic entirely and close to clients.

7.5.2 CoMon: Caching Efficiency Gain and Overhead

We performed an extensive simulation study to evaluate both the caching efficiency gain of CoMon and its overhead. In particular, we use the server hit ratio (as defined in Section 7.1) to evaluate the caching efficiency⁹. Regarding the overhead, we evaluate both (i) the hop count overhead of MAR and FTBM, and (ii) the signalling overhead of control messages.

⁹ Note that CoMon maximizes the cache diversity and provides centralized control on content replacements by design (Section 7.4). Therefore, we exclude those two metrics from our evaluation.

Simulation Setup

We simulated using ccnSim [CRR13], a highly scalable OMNeT++ based [OMN] chunk-level simulator for NDN. ccnSim is primarily developed for caching behaviour, and it offers several caching-related functionalities. It also implements several cache management solutions. We adapted ccnSim to produce the behaviour of CoMon. As baselines for comparison, we also simulated the original NDN system and two notable prior solutions, namely: (i) probabilistic caching (ProbeCache) [PCP12] and (ii) Leave Copy Down (LCD) [LCS06] (Section 7.2).

In each simulation run, 10^6 interest packets are generated from all clients. Arrival of these packets is modelled as Poisson process as in realistic workloads [MSB⁺15]. In order to increase the competition among the contents on the available storage, we used small cache sizes with a uniform size of $c = 10$ contents and a relatively large content population $P = 100 \times C$ (*i.e.* 100 times the overall cache space). Contents are permanently stored at servers that are located outside the network, where each server is accessible via only one gateway node selected randomly at the beginning of each simulation run. Following results of widely accepted measurement studies (*e.g.* [CKR⁺07, HS08, BW02]), we modelled content popularity as both Zipf distribution and Mandelbrot-Zipf (MZipf) distribution¹⁰ [Sil99]. However, since there is no consensus in the literature on the distribution parameters, we performed simulations with varying values of $\alpha \in \{0.5, 0.8, 1, 1.5, 2\}$. In addition, we also simulated using MZipf's parameters: $\alpha = 0.8$, $q = \{5, 50\}$. Note that with $\alpha \leq 1$ (and large q) we catch very bad cases of unpopular contents, since the smaller α (and the larger q) the larger the number of contents that represent the majority of client requests.

We simulated with a uniform data packet size of 1100 bytes. We selected such a small size (even smaller than Ethernet's maximum transfer unit) in order to not bias the signalling overhead results (which are normalized over the size of data packets). We set the observation window and FoC's threshold to 60 seconds and 0.2, respectively. In a real network, however, both parameters should be adjusted according to the network size and dynamically to the observed traffic volume. We also selected as few as top [5%] PRCS-ranked nodes as MNs.

Server Hit Ratio

Figure 7.6 plots the server hit ratio (SHR) results of CoMon, NDN, LCD, and ProbeCache in the three ISP network topologies with MAR-V1. These results represent the minimum values (*i.e.* best SHR) achieved with NDN, LCD, and

¹⁰ MZipf distribution models the probability of accessing an item at rank i out of P items as $p(i) = K/(i+q)^\alpha$, where $K = \sum_{i=1}^P 1/(i+q)^\alpha$, α is the distribution skewness, and $q \geq 0$ is the so-called plateau factor. Zipf distribution is a special case of MZipf distributions with $q = 0$.

ProbeCache and the maximum values (*i.e.* worst SHR) achieved with CoMon, over 20 simulation runs. The results with MAR-V2 are very similar.

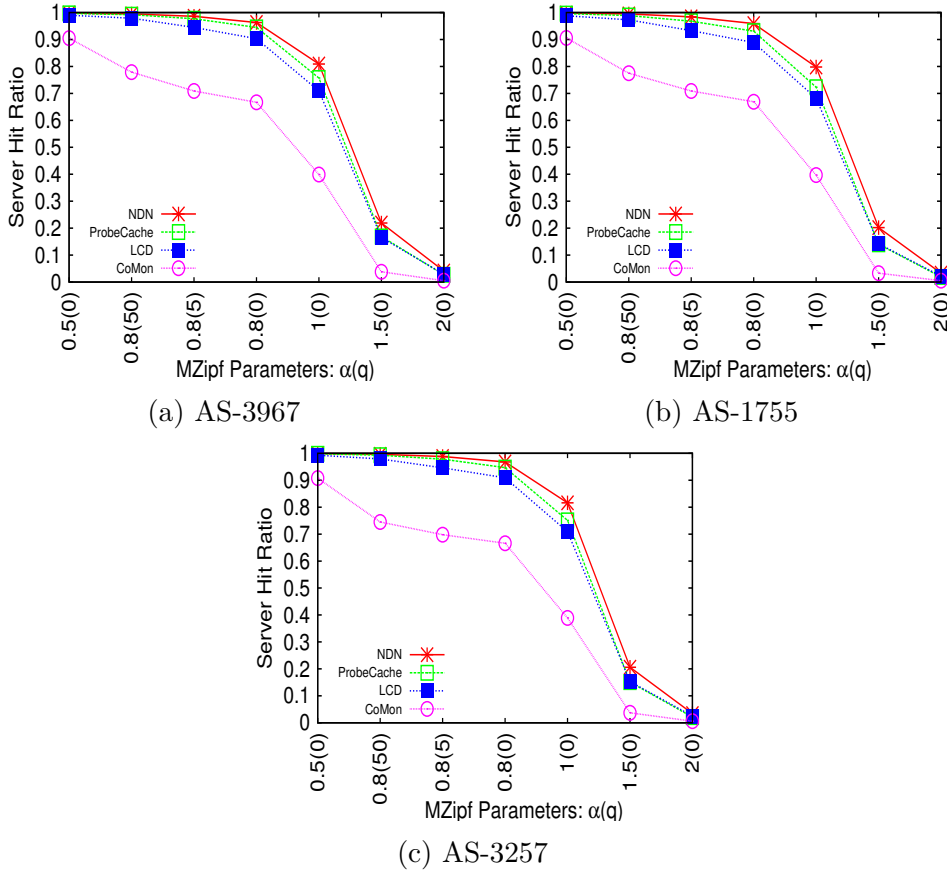


Figure 7.6: Server hit ratio: CoMon vs. vanilla NDN, LCD, and probabilistic caching

We can see that CoMon significantly reduces the SHR in comparison to the three other systems. This means that, with CoMon, the network traffic that crosses other ASs (thus the transit costs) also decreases significantly. For instance, in AS-3967 CoMon reduces the SHR of LCD (the best among the three other systems) by about 30% for $\alpha = 0.8$. Although the SHR results in the four systems are better with other values α (*i.e.* less or more skewed popularity distribution), CoMon is always the superior with a remarkable difference, except for very highly skewed popularity distribution ($\alpha = 2$). The SHR improvement achieved with CoMon is relatively low at $\alpha = 0.5$ since in this case the majority of interest packets is associated with different content names (*i.e.* benefit of caching is low in general). In contrast, at $\alpha \geq 1.5$ all systems perform well (even the vanilla NDN). This is because the majority of interest packets correspond to a very small fraction of contents, and it hence is consumed either by caches or PITs. That is to say, utility of caching is low in this case. The improvements that are achieved with CoMon in AS-1755 and AS-3257 are very similar.

The superiority of CoMon confirms the utility of global cache coordination (except when popularity is highly skewed). It enables to take efficient caching-related decisions and to perform cache-aware routing, thus to maximize the number of interest packets consumed by caches.

Next, we evaluate three overhead components: (i) the hop count overhead of MAR, (ii) the hop count overhead of FTBM, and (iii) the signalling overhead of control messages used for coordination.

Hop Count Overhead of MAR

We first measure the hop count overhead of MAR comparing three cases: (i) when MAR is disabled, (ii) when MAR-V1 is enabled, and (iii) when MAR-V2 is enabled. Figure 7.7 plots the CDF of the hop count distribution for the three cases. In order to distinguish the hop count overhead of MAR from that of FTBM, we consider the interest packets that are consumed by origin content provides (*i.e.* those which travelled the entire path till gateway nodes). The plotted values of the first two cases represent the simulation run with maximum average hop count (among 20 runs), while the plotted values of the third case represent the simulation run with minimum average hop count. Table 7.5 reports the corresponding statistics.

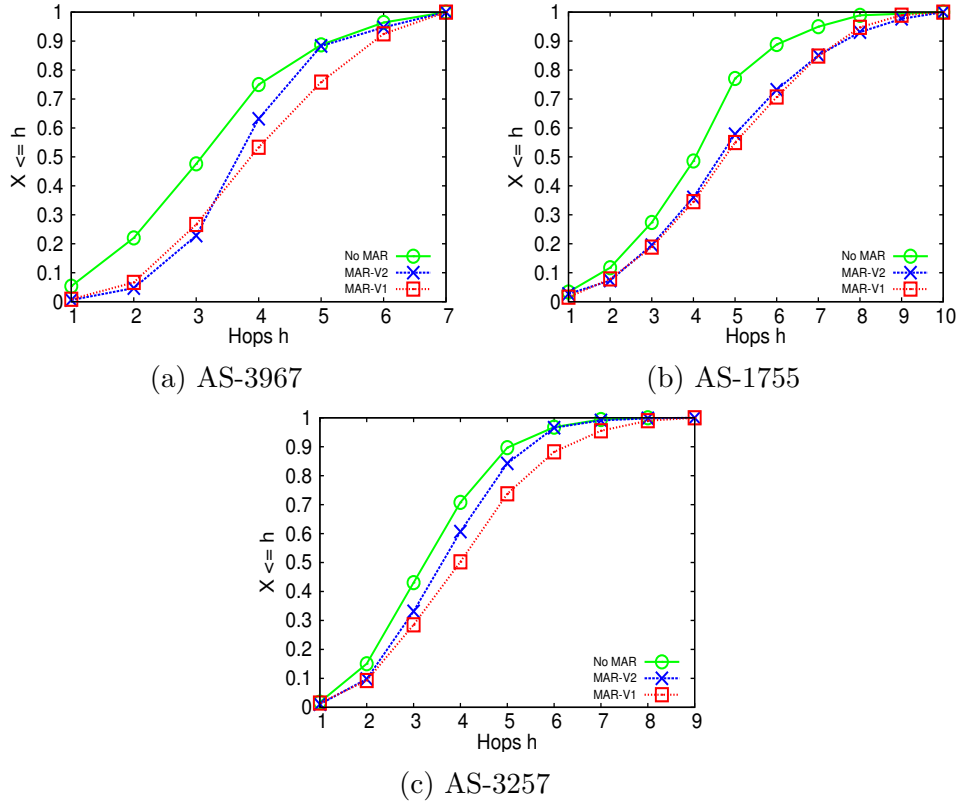


Figure 7.7: Hop count overhead of MAR

Table 7.5: Hop count statistics of MAR

		No MAR	MAR-V2	MAR-V1
AS-3967	Minimum	1	1	1
	Maximum	7	7	7
	Median	4	4	4
	Average	3.65	4.26	4.44
	Std. dev.	1.44	1.15	1.38
AS-1755	Minimum	1	1	1
	Maximum	10	10	10
	Median	5	5	5
	Average	4.54	5.38	5.38
	Std. dev.	1.66	1.94	1.93
AS-3257	Minimum	1	1	1
	Maximum	8	9	9
	Median	4	4	4
	Average	3.83	4.16	4.54
	Std. dev.	1.32	1.33	1.62

The results in general show that the hop count overhead of MAR is small, and that MAR-V2 slightly outperforms MAR-V1. More precisely, the median hop count values are the same in the three cases, while the average hop count increases only slightly when MAR is enabled. For instance, in AS-3967 the average hop count increases from 3.65 to 4.26 and 4.44 with MAR-V2 and MAR-V1, respectively. Similar hop count overhead is caused by MAR in the two other network topologies.

We attribute the results above (*i.e.* low hop count overhead caused by MAR) to the PRCS' strategy of giving preference for nodes that are located close to clients. As for the superiority of MAR-V2 over MAR-V1, it is attributed to MAR-V2's strategy of selecting routes of interest packets according to the requested content name (rather than simply forwarding packets directly to the closest MN).

Hop Count Overhead of FTBM

We now evaluate the hop count overhead of FTBM. As discussed in Section 7.4.3, this overhead affects only interest packets that are consumed by caches or PITs before they are monitored. Therefore, we measure this overhead when content popularity is modelled as Zipf distribution with $\alpha = 2$. In this scenario, as we discussed above, the majority of interest packets are consumed by caches or PITs.

Figure 7.8 plots the hop count distribution both when FTBM is disabled and when it is enabled. More precisely, to capture the highest overhead, the plotted values with FTBM represent the simulation run with maximum average hop count, while the values without FTBM represent the run with minimum average hop count. We summarize the corresponding statistics in Table 7.6.

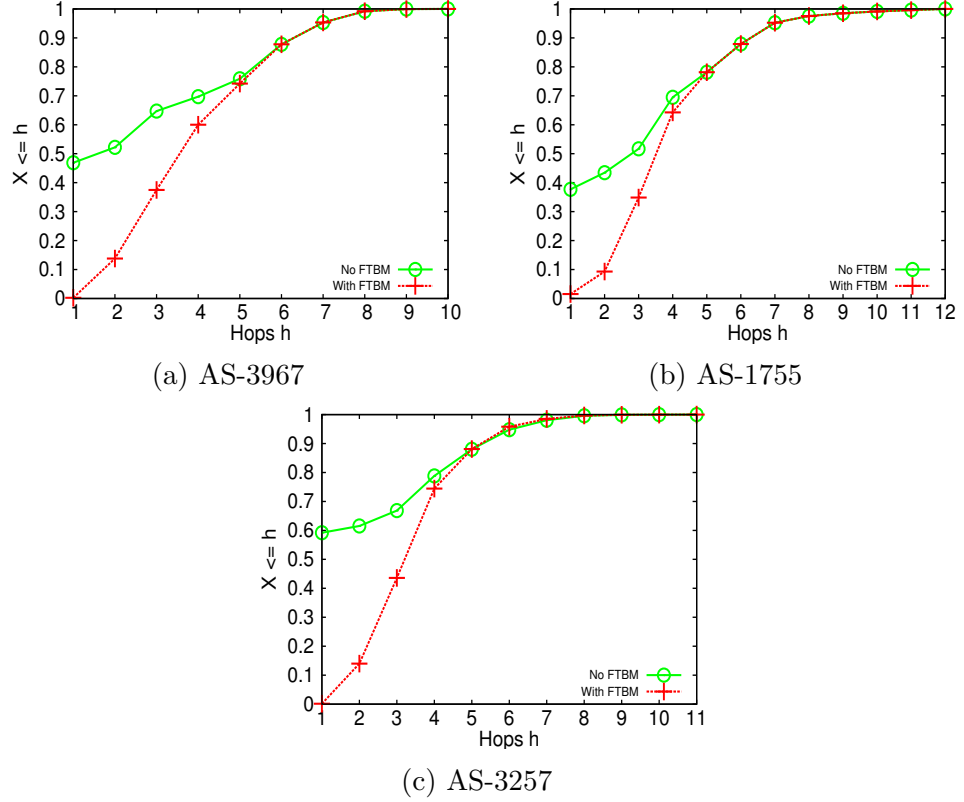


Figure 7.8: Hop count overhead of FTBM

The results show that FTBM causes low to intermediate hop count overhead. Comparing the average values, we can see that the hop count overhead ranges from 27.2% (increased in AS-1755 from 3.46 to 4.40) to 52.6% (increased in AS-3257 from 2.53 to 3.86). Recall that we measured this overhead with highly skewed content popularity distribution (as can be seen in Figure 7.8, 37.7% to 59.2% of interest packets were consumed after one hop only). The hop count overhead of FTBM was lower with less skewed content popularity distributions.

Signalling Overhead

We measure the signalling overhead of coordination by normalizing the total number of bytes of control messages (the three message types in Figure 7.2) over the total number of bytes of regular data packets. Figure 7.9 plots the results for the three network topologies. The plotted values represent the maximum

Table 7.6: Hop count statistics of FTBM

		No FTBM	With FTBM
AS-3967	Minimum	1	1
	Maximum	10	10
	Median	2	4
	Average	3.08	4.32
	Std. dev.	2.40	1.69
AS-1755	Minimum	1	1
	Maximum	12	12
	Median	3	4
	Average	3.46	4.40
	Std. dev.	2.42	1.72
AS-3257	Minimum	1	1
	Maximum	11	11
	Median	1	4
	Average	2.53	3.86
	Std. dev.	2.08	1.34

signalling overhead observed in 20 simulation runs.

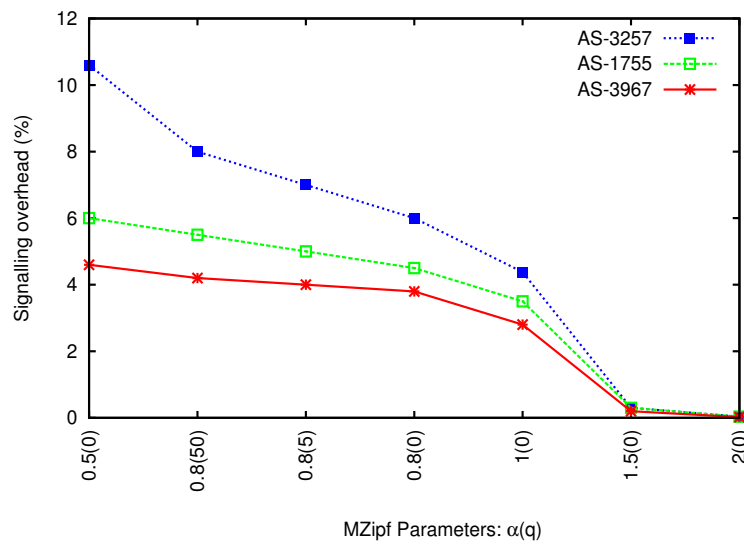


Figure 7.9: Signalling overhead of control messages

Altogether, the results show that the signalling overhead incurred by control messages is very small. The worst result is associated with AS-3257 at $\alpha = 0.5$, with a normalized signalling overhead of about 11% only.

More generally, the signalling overhead increases with the network size and with the plateau factor (q), and it decreases with the popularity skewness parameter (α). These results are to be expected. In particular, the overhead increases with the network size because the number of MNs also increases with the network size. As such, the volume of information exchanged between the AC and MNs (messages labelled "1" and "3" in Figure 7.2) increases almost linearly with the number of MNs. In addition, the overhead increase with q and decreases with α because the number of observed contents (the main factor that determines the size of control messages labelled "1") also increases with q and decreases with α .

To sum up, both the hop count overhead of MAR and the signalling overhead of coordination are low. FTBM also results in low to intermediate hop count overhead. Nevertheless, we argue that the achieved caching efficiency gain (represented by a remarkable SHR improvement) pays off this overhead.

7.6 Summary

We presented in this chapter a coordination framework for NDN called CoMon. CoMon fulfils its preset design requirements (Section 7.3.1). Table 7.7 summarizes the design elements and features that enable CoMon to fulfil these requirements.

Table 7.7: CoMon (coordinated caching): Mapping design elements to the requirements

	R1	R2	R3
The AC takes decisions in a centralized way	✓		✓
The AC has network-wide information	✓		
MAR & FTBM enable for full traffic coverage	✓		
The AC shares the global cache state with MNs		✓	
PRCS gives preference for nodes close to clients			✓
Monitoring and cache-aware routing are performed by a small number of nodes			✓
The FoC technique reduces the frequency of updates			✓
Each packet is checked only once			✓

Through simulations, we showed that CoMon realizes effective, yet lightweight, coordination on AS-wide scale. In particular, we employed CoMon for coordinating caching decisions. CoMon, incurring very low signalling overhead and low to intermediate hop count overhead, enables to achieve a remarkable caching efficiency improvement in comparison both to the vanilla NDN and to prior solutions.

We show in the next chapter that the design concepts of CoMon can be applied to address another problem in NDN: the detection and mitigation of a new type of DDoS attack known as the Interest Flooding Attack (IFA).

Coordinated Defence Against Interest Flooding in NDN

The work that we present in this chapter is stimulated from a systematic vulnerability of a new technology that is proposed to replace the current Internet. In particular, we focus on an NDN-tailored DDoS attack coined in the literature with the name Interest Flooding Attack (IFA). It has been experimentally shown that IFA is relatively easy and cheap to perform, and that it is able to degrade the QoE of users significantly [CCGT13, AMM⁺13]. The attack aims at flooding the network and obstructing the service received by legitimate users. Towards this end, the adversary misuses two design properties of NDN: (i) routing based on longest name-prefix match and (ii) storing a forwarding state per interest packet in the Pending Interest Table (PIT). In more detail, the adversary sends interest packets with unique fake content names targeting name space(s). Consequently, one PIT entry is created per interest packet in each NDN node on the path. These entries stay in the PITs till they expire at the end. Succeeding to overload some or all PITs results in dropping legitimate interest packets.

Despite the considerable amount of research on IFA, proposed defence mechanisms (*e.g.* see [GTUZ13, CCGT13, AMM⁺13, GCFE13, DWFL13] and the references therein) have one or more of the following drawbacks: First, attack detection is difficult close to sources especially for distributed low-rate IFAs. This is because the observable amount of traffic is relatively small. In contrast, detection close to the attack targets is likely not robust due to the large volume of attack traffic. Second, legitimate traffic can be damaged because proposed reactions do not distinguish legitimate packets from malicious ones. Third, every node is required to perform attack detection and mitigation. Consequently, independent or not well coordinated decisions can result in inaccurate attack detection, overreactions, or inequitable punishments [AMM⁺13]. Collaborative mechanisms also result in high signalling overhead.

The main contribution that we present in this chapter is a new defence mechanism against IFA. The mechanism detects and mitigates IFAs in a coordinated way. This is done based on aggregated and timely knowledge of content access information and forwarding states of interest packets.

In practice, we adapt CoMon, our framework for coordination in NDN. This choice is motivated by CoMon's ability to realize coordination that is efficient and feasible. This was showed in Chapter 7 in which we developed CoMon initially to coordinate caching-related decisions in NDN on AS-wide scale. Using the design concepts of CoMon, IFAs are detected and mitigated in our defence mechanism by the monitoring nodes (MNs) with the aid of the centralized AS controller (AC).

8.1 Interest Flooding Attack

Despite the built-in security features of NDN (Section 6.7), its design opens the door for new types of attacks [AHZ15]. Among those, the Interest Flooding attack (IFA) is harmful, and it is relatively easy to perform with rather limited resources. IFA can be mounted by taking advantage of two NDN's properties: (i) storing a forwarding state per interest packet in each node on the path and (ii) routing by longest name-prefix match.

In practice, the adversary (through distributed bots) produces a large number of interest packets and inserts them into the network. With such an attack, the adversary aims at (i) overloading PITs of NDN nodes and/or (ii) immersing the targeted content provider so they cannot handle legitimate interest packets. Note that since contents in NDN are requested by their names (rather than IP addresses), it is difficult to attack specific hosts or nodes in the network. Instead, the adversary can easily target content providers through their name spaces.

IFAs are classified, according to the interest packets used in the attacks, into three types [GTUZ13]. In particular, interest packets can be used to request (i) existing, (ii) dynamically-generated, or (iii) non-existent content. Caches of NDN nodes provide a built-in mitigation for type (i). The attack in type (ii) aims both at exhausting resources of origin content providers on serving malicious interest packets as well as at consuming nodes' PITs. However, its impact on PITs is eliminated once the corresponding data packets are received. In type (iii), as can be seen in the example in Figure 8.1, the adversary targets a specific name space¹ (*e.g.* `"/de.tu-darmstadt/"`), by producing interest packets with name-prefixes belonging to the targeted name space (*e.g.* `"/de.tu-darmstadt/dissertations/"`) appended by a unique suffix per packet. The name suffixes are chosen such that the resulted content names are fake, *i.e.* they will not be satisfied. Consequently,

¹ This way, the malicious interest packets are routed towards and as close to the origin data provider (*i.e.* the victim) as possible, which increases the effectiveness of the attack [AMM⁺13].

a PIT entry is created per fake interest packet in each node on the path. Note that the total number of unique fake interest packets sent during the attack represents an upper bound on the amount of memory that can be exhausted in the PITs of victim nodes. Note also that the closer the node to the content provider responsible for the targeted name space, the more the fake interest packets that the node receives, thus the larger the impact of the attack on its memory. Since malicious interest packets are non-satisfiable, the corresponding PIT entries remain till they eventually expire. Due to the aforementioned consequences of type (iii), it is considered the most harmful type of IFA. We focus on this type only, and we associate it with the name IFA from now onwards.

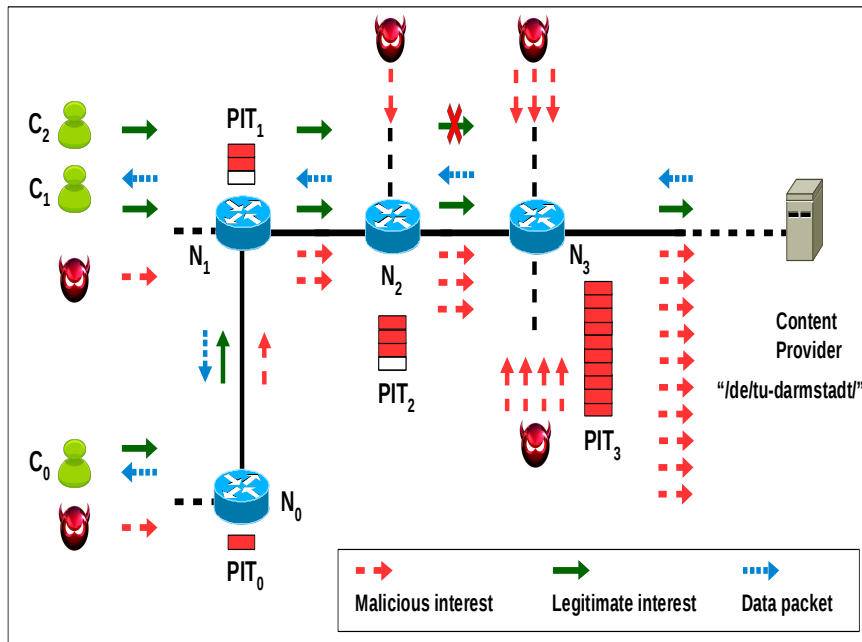


Figure 8.1: Example of IFA: All interest packets use the same name-prefix *"/de.tu-darmstadt/"*, thus are routed towards the same server. At the beginning, C_0 and C_1 concurrently issue one interest packet each, using the same suffix. Hence, both packets are aggregated, *i.e.* they hold a single PIT entry in each bypassed node. The created entries are eliminated once the corresponding data packet arrives. After that, the five adversaries issue malicious interest packets (10 in total) almost concurrently. Each of those packets uses a unique non-existent suffix. Consequently, a single PIT entry is created per malicious interest packet in each bypassed router, staying till it expires. Next, before the entries that correspond to the malicious interest packets expire, C_2 issues a legitimate interest packet. Assuming a PIT capacity of 10, this packet is dropped by N_3 because N_3 's PIT is full.

8.2 Related Work

We restrict the discussion in this section to prior work on IFA. For a broad overview of research on security of NDN and other ICN architectures, the reader is referred to [AHZ15].

IFA was discussed for the first time by Lauinger [Lau10]. After that, Gasti et al. [GTUZ13] detailed IFA's operation and types. Both studies suggested tentative defence mechanisms against IFA. Evaluations of those mechanisms, however, were left for future research. Afterwards, several studies evaluated the effectiveness of IFA, and they agreed on the necessity to protect nodes and content providers in NDN against such an attack. These studies also proposed and evaluated several defence mechanisms against IFA. The proposed mechanisms can be classified as either autonomous or collaborative.

On the one hand, in the autonomous mechanisms, each individual node detects attacks based on its local view of network traffic and/or PIT usage. For instance, an attack is detected when the observed ratio of unsatisfied interest packets (or PIT expiration rate) exceeds a preset threshold. Then, a reaction (*e.g.* dropping part of suspicious incoming interest packets) is taken by each node independently. Such mechanisms, although being simple and inexpensive, have three key drawbacks: First, attack detection is difficult close to attack sources, because the amount of received traffic (thus the effect of the attack) is small. This is particularly true for distributed low-rate IFAs. Late detection and reaction, in contrast, take place after wasting lots of resources in several regions of the network, and likely cannot prevent the attack before it causes a big damage. Secondly, independent attack reactions may result in overreactions or inequitable punishments [AMM⁺13]. The third drawback is that applied detection algorithms do not distinguish malicious interest packets from legitimate ones. Hence, they may harm legitimate traffic.

Notable examples of autonomous mechanisms include [GCFE13], satisfaction-based acceptance [AMM⁺13], and the autonomous version of Poseidon [CCGT13]. Widjaja [Wid12] presented another autonomous mechanism with a unique approach. In particular, the author proposed to defend against IFAs by removing the PIT completely from the node model. Instead, nodes cache interest packets in the cache store as regular data packets. Such a mechanism, however, makes indexing and forwarding of interest packets more complicated, and it hence was considered impractical [WZQZ14].

In the collaborative countermeasures, on the other hand, nodes exchange information about their local observations and taken reactions. This information is used by the nodes to update parameters related to attack detection (*e.g.* some thresholds) and reaction decisions (*e.g.* percentage of dropped interest packets). By this, the nodes can detect and mitigate attacks faster (*i.e.* while they are in progress) and close to their sources. This applies for Interest Traceback [DWFL13]

and pushback-like mechanisms such as satisfaction-based pushback [AMM⁺13], Cooperative-Filter [WZQZ14], and the collaborative version of Poseidon [CCGT13]. These mechanisms, although outperform their autonomous counterparts, still suffer from one or more of the aforementioned drawbacks. Furthermore, they incur high overhead.

8.3 Design Requirements and Overview

8.3.1 Design Requirements

We aim to achieve effective defence against IFAs that can overcome the drawbacks of prior defence mechanisms. Towards this end, we guide the design of our defence mechanism by the following requirements:

- R1) IFAs should be detected based on aggregated (can be network-wide) information of packets transmitted and corresponding forwarding states. Such information enables accurate attack detection, even for low-rate IFAs.
- R2) Both the detection and the mitigation of IFAs should be performed at an early stage close to the attack sources (before malicious interest packets consume lots of resources).
- R3) Both duplicate attack detections and overreactions should be avoided.
- R4) The mechanism should be able to distinguish between legitimate interest packets and malicious ones to avoid damaging legitimate traffic.
- R5) The mechanism should induce a low overhead.

8.3.2 Architecture and Operation Primitives

Our goal is to realize a defence mechanism against IFA that fulfils all above requirements on AS-wide scale. However, as we discussed in Chapter 7, the distributed nature of ASs in addition to the huge volume and high dynamics of coordination-related information (*i.e.* information to be aggregated and disseminated) render such a solution impractical.

We addressed a similar challenge in Chapter 7 (coordinating caching-related decisions on AS-wide scale) by CoMon, a framework for coordination in NDN. Motivated by CoMon’s results in which we showed its ability to realize effective and lightweight coordination, we decided to use its design concepts as a basis for our defence mechanism against IFA.

By adapting CoMon, our solution relies on a small number of NDN nodes to monitor network traffic and forwarding states of interest packets. Attacks are

detected either by the MNs with the aid of the AC (which aggregates observations of all MNs). Reaction to potential attacks is then performed by the MNs. Similar to the original CoMon framework, MNs are selected such that the entire network traffic passes (or is enforced to pass) through at least one of them, at an early stage. This way, their unique observations jointly represent the entire network, and they can detect and react to attacks quickly.

More particularly, our solution (as in the original CoMon framework) is designed to work within an AS. Each AS network consists of a set V of nodes. We use the same principal elements of CoMon (Figure 8.2): an AC, NNs (NDN Nodes), and MNs.

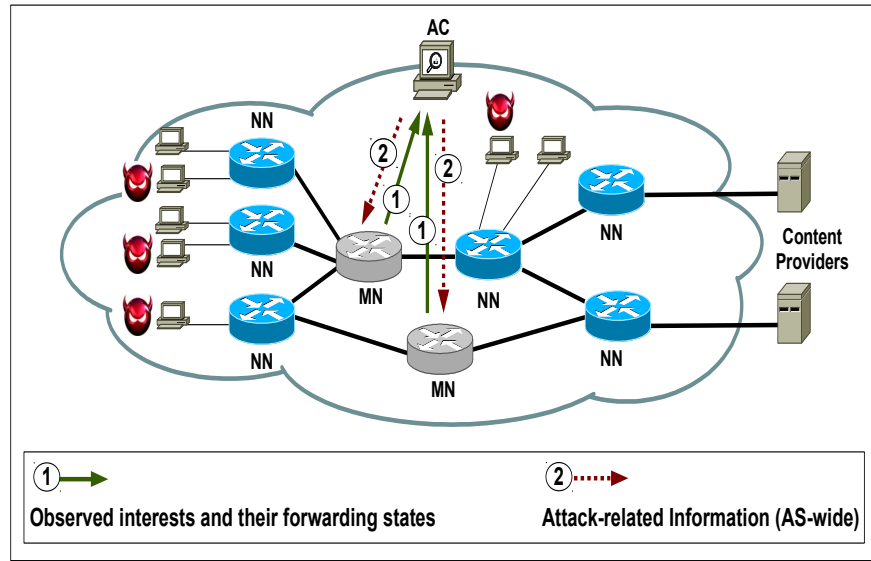


Figure 8.2: System architecture and the control messages used in the coordinated defence against IFAs: "AC" stands for AS Controller, "NN" stands for NDN Node, and "MN" stands for Monitoring Node

We describe now how CoMon is adapted to provide a coordinated defence against IFAs:

1. The AC collects monitoring reports about exchanged packets and corresponding forwarding states from a predetermined set of MNs. It uses this information to detect attacks on AS-wide scale. After that, the AC shares attack-related information with the MNs.
2. NNs are similar to standard NDN nodes. They mainly perform routing and caching tasks only.
3. MNs (a set $M \subset V$), in addition to the functions of regular NDN nodes, monitor interest packets passing through them, and they check whether these packets are satisfied (by data packets) or not. Consequently, the

MNs compute expiration rates of their PIT entries per incoming interface. When the expiration rate exceeds some preset threshold, the corresponding incoming interface(s) and name-prefix(s) are considered infected.

The MNs periodically report summaries of their observations and results to the AC. The AC in turn sends the MNs a feedback summarizing information of attacks ongoing over the entire AS. The MNs also are responsible for mitigating potential IFAs.

8.4 Design Specifications

We describe in this section the design specifications of our solution that are not discussed in Section 8.3.2.

8.4.1 Detection of IFAs

IFAs are basically detected by each MN independently. The corresponding detection algorithm can be outlined as follows:

1. MN m continuously monitors the utilization of its PIT, arriving interest packets, and corresponding PIT entries. As mentioned in Section 7.4.1, the MNs do not monitor previously monitored interest packets (*i.e.* when the flag *checked* = 1). This way, duplicate detection is avoided in our mechanism.
2. MN m calculates the PIT utilization ratio $U(m, q)$. This ratio represents the maximum number of PIT entries observed during the observation window q , divided by the PIT size.²
3. MN m calculates the PIT expiration rate $E(m_f, q)$ per incoming interface f over the observation window q , as follows:

$$E(m_f, q) = \frac{e(m_f, q)}{e(m_f, q) + s(m_f, q)} \quad (8.1)$$

where $e(m_f, q)$ and $s(m_f, q)$ represent the corresponding counts of expired and satisfied PIT entries, respectively. As mentioned above, both values consider only PIT entries that correspond to interest packets not monitored before (*i.e.* the value of *checked* = 0).

² The observation window here refers to the time period after which the MNs (i) check existence of attacks (*i.e.* triggers the detection function) and (ii) send monitoring reports to the AC.

The two detection parameters above are similar to the ones which were used in [WZQZ14]. However, aiming to achieve better detection, our detection algorithm employs them differently. In particular, our mechanism computes them based on aggregate data and without repetitions.

4. MN m triggers the reaction function at the end of q if $E(m_f, q)$ is positive and $U(m, q)$ exceeds a threshold τ . In this case, m identifies the name-prefixes of expired PIT entries. Those name-prefixes as well as the interface f are then considered infected. The MN then calculates the PIT expiration rate $E(m_f^j, q)$ for each infected name-prefix j (calculated as in Eq. 8.1, but only for entries with prefix j). Next, m sends a report to the AC including its observations and detection results (Figure 8.2).

To minimize false positives, due to transient failures in accessing the network or delivery of packets, τ should be given a large value. However, such a setting reduces the sensitivity of the detection function for low-rate IFAs. We address this problem by implementing an additional, AS-wide, detection in our defence mechanism performed by the AC. The corresponding algorithm consists of the following steps:

1. The AC aggregates the monitoring information that it received from the MNs at the end of the observation window q .
2. For each name-prefix j , the AC calculates the ratio $Q(j, q)$ which represents the maximum number of corresponding expired PIT entries divided by the PIT space. $Q(j, q)$ considers only PIT entries corresponding to interest packets that are monitored first by the MN which sent the report. This way, $Q(j, q)$ represents an upper bound on the PIT entries that can be occupied in any subsequent node by interest packets with name-prefix j .
3. If $Q(j, q)$ exceeds a threshold γ , j is considered infected.
4. The AC informs the MNs about the infected name-prefixes. Each MN, in turn, calculates the expiration rate for each of those name-prefixes on all incoming interfaces and then triggers the reaction function accordingly.

8.4.2 Reaction Against Potential IFAs

When triggered, the reaction lasts along the next observation window. The pseudo-code of the reaction function is provided in Algorithm 6.

In essence, the PIT expiration rate is used to determine the probability of rejecting (*i.e.* dropping) incoming interest packets. We adapted this idea from the satisfaction-based acceptance mechanism [AMM⁺13], motivated by its simplicity.

Algorithm 6 Reaction against potential IFAs

```

1:  $J \leftarrow$  infected name-prefixes observed on interface  $f$  during observation window  $q$ 
2: procedure REACTION( $f, J$ )
3:   while receiving interest packets on  $f$  do
4:     for each interest packet  $I$  do
5:       if  $checked = 0 \ \& \ I \in J \ \& \ I \notin A$  then            $\triangleright A$ : Satisfied interests  $\in J$ 
6:         Drop  $I$  with probability  $P(E(m_f^j, q))$             $\triangleright P(a) = a \ (\forall a \in [0, 1])$ 
7:       end if
8:     end for
9:   end while
10: end procedure

```

Our solution, however, is more generic as it can incorporate any other reaction strategy. The function uses a uniform probability distribution model, and it is applied in our algorithm on each infected interface per name-prefix (line: 6), while [AMM⁺13] disregards the name-prefix granularity.

Before applying the strategy above, the algorithm first (line: 5) excludes (*i.e.* directly accepts) interest packets that are (i) checked earlier by another MN, (ii) not belonging to an infected name-prefix, or (iii) satisfied earlier (*i.e.* returned a data packet). These exceptions aim at fulfilling the design requirements R3 and R4 (Section 8.3.1). In particular, by directly accepting previously monitored interest packets (exception (i)), duplicate detection and overreactions are avoided. As for exceptions (ii) and (iii), they are included to avert dropping legitimate interest packets. More precisely, both interest packets that do not belong to infected name-prefixes (exception (ii)) as well as those that were satisfied earlier (exception (iii)) should be accepted directly since they are surely not part of an attack.

8.5 Evaluation

We performed a simulation study to evaluate both the effectiveness of our defence mechanism and its signalling overhead. We first describe the setup and evaluation parameters. After that, we discuss the results.

8.5.1 Simulation Setup

ccnSim (the simulator that we used in Chapter 7 for implementing and evaluating CoMon) is developed foremost to study caching efficiency, and it implements PIT and content names in an elementary way. Hence, ccnSim (without substantial changes) is not suitable to simulate neither IFA nor defence mechanisms. Instead, we decided to implement both IFA and our defence mechanism in ndnSIM [AMZ⁺12], an implementation of NDN for the NS-3 network simulator [NS3].

ndnSIM implements most NDN design elements as they are described in [JST⁺09], and it was used in many related studies. We also used ndnSIM to simulate the satisfaction-based pushback (SBP) mechanism [AMM⁺13] (which was considered an effective defence mechanism).

We fed the simulator with the three ISP network topologies that we described in Section 7.5. At the beginning of each simulation run, $\lceil 70\% \rceil$ of the nodes are selected as consumer nodes and three of the rest are selected as gateway nodes. Attackers accessed the network through $\lceil 25\% \rceil$ of consumer nodes. Those selections were performed uniformly at random. Each legitimate client requests existing contents with a rate of 100 interest packets per second (ipps). In contrast, each attacker requests non-existent contents at higher rates. In particular, we experimented with several attack rates ranging from 200 ipps to 20000 ipps. However, the influences of attack rates lower than 300 ipps were very small. As for attack rates above 5000 ipps, their influences were nearly the same without defence: the large majority of legitimate interest packets were dropped. Therefore, we show and discuss only the results of two attack rates representing two main cases: 500 *ipps* (low) and 10000 *ipps* (massive).

We used a uniform PIT size of 5000 entries, a PIT timeout period of 2 seconds, and a uniform data packet size of 1100 bytes³. Each simulation run lasted for 540 simulation seconds. The attack started at the beginning of the 61st second and stopped at end of the 360th second. This duration was long enough to examine both the influence of the attack and the effectiveness of the defence mechanisms.

We configured the observation window in the MNs to 10 seconds. In a real network, this value should be adjusted according to the network size, as well as dynamically to the observed traffic volume and attack rate. As we discussed in Section 8.4.1, τ and γ (both $\in (0, 1]$) should be assigned relatively small and large values, respectively. In our experiments, we set τ to 0.3 and γ to 0.5. The simulator selected the top $\lceil 10\% \rceil$ PRCS-ranked nodes as MNs.

We repeated each experiment 20 times and obtained nearly the same results from repeated runs. We plot the results of one exemplary simulation run for each experiment, and we also provide the main statistics of the corresponding 20 runs.

8.5.2 Evaluation Metrics

We use the following three metrics to evaluate our solution. The first two measures the effectiveness against IFAs, while the third measures the signalling overhead.

1. Satisfaction ratio of legitimate interest packets: This metric is an indicator for the QoE of legitimate clients during the attack period.

³ As mentioned in Section 7.5.2, we selected such a small packet size to not bias the signalling overhead results.

2. Network-wide PIT usage: It refers to the utilization ratio of the overall PIT space. This metric is an indicator for the impact of the attack on the usage of PIT (the main resource targeted by IFA).
3. Signalling overhead: It is measured by normalizing the number of bytes used by defence-related messages (Figure 8.2) over the number of bytes used by regular data packets.

8.5.3 Results

Effectiveness Against IFAs

We plot in Figure 8.3 the satisfaction ratio of legitimate interest packets in the three topologies, under each of the two aforementioned attack rates.

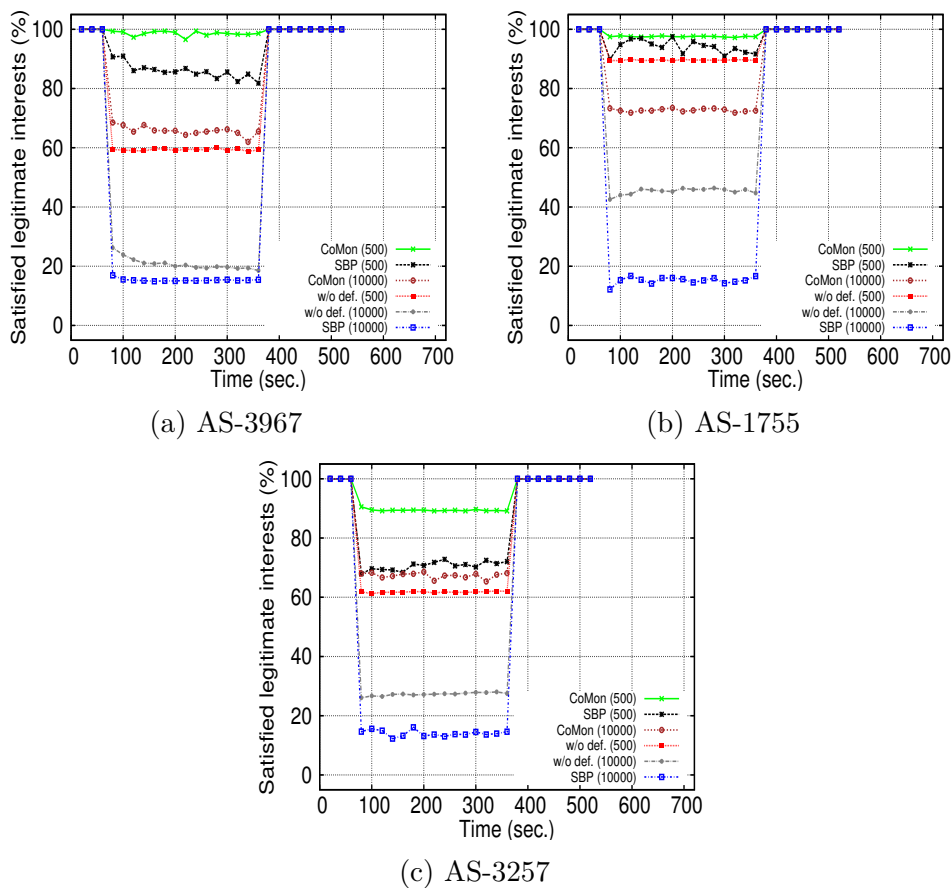


Figure 8.3: Satisfaction ratio of legitimate interest packets under two attack rates (attack period: seconds 61 – 360)

A defence mechanism is considered effective if it can remarkably raise the satisfaction ratio of legitimate interest packets, while the network is under attack. We compare the results of three systems: (i) the vanilla NDN system (*i.e.* without

defence), (ii) NDN with our defence mechanism, (iii) NDN with satisfaction-based pushback (SBP) [AMM⁺13]. We plot the reported satisfaction ratio of legitimate interest packets before the attack, during the attack (second 61 – second 360), and after the attack. The satisfaction ratios were reported every 20 seconds. Table 8.1 summarizes the main related statistics during the attack period calculated over 20 simulation runs.

Table 8.1: Satisfaction ratio of legitimate interests during attack periods

		SBP		CoMon	
		Attack rate (ipps)		Attack rate (ipps)	
		500	10000	500	10000
AS-3967	Minimum	80.47	14.53	93.64	60.40
	Maximum	90.92	18.29	99.68	69.37
	Median	85.64	16.37	96.40	65.49
	Average	85.54	16.37	96.52	65.31
	Std. dev.	2.83	1.08	1.86	2.94
AS-1755	Minimum	87.91	13.90	94.24	66.45
	Maximum	97.49	16.71	99.78	73.77
	Median	90.25	15.01	97.12	70.04
	Average	90.38	15.07	96.97	70.09
	Std. dev.	1.56	0.72	1.65	2.31
AS-3257	Minimum	67.94	11.65	85.33	62.43
	Maximum	74.44	17.21	91.87	69.29
	Median	71.98	14.25	88.46	66.57
	Average	72.01	14.37	88.40	66.32
	Std. dev.	1.46	1.62	2.07	2.26

The improvement which is achieved with our defence mechanism is obvious: during the attack period, legitimate interest packets achieve much higher satisfaction ratio with our mechanism. The highest improvement can be seen in the AS-3967 network topology: the average satisfaction ratio raises with our mechanism from about 59% to about 96% under a low attack rate, and it raises from about 23% to about 65% under a massive attack rate. In contrast, the average satisfaction ratios that are achieved with SBP under the low and massive attack rates are only about 86% and 16%, respectively. Although the values are slightly different in the other two network topologies, the conclusion above holds for them: our mechanism significantly improves the satisfaction ratios of legitimate interest packets and remarkably outperforms SBP. Note that the impact of SBA

under massive attack rates becomes negative! This is likely because SBP does not distinguish between malicious and legitimated interest packet.

The effectiveness of our mechanism against IFAs is also confirmed by the results of the second metric: Figure 8.4 shows that our defence mechanism lowers the global (*i.e.* network-wide) PIT usage during the attack period. For instance, the PIT usage in the AS-3967 network topology is reduced from about 16% to about 11% under attack rate of 500 ipps and from about 36% to about 29% under attack rate of 10000 ipps. Similar reductions (although with slightly lower values in some cases) can be seen in the results of the other two network topologies. Table 8.2 summarizes the main related statistics for the corresponding 20 simulation runs.

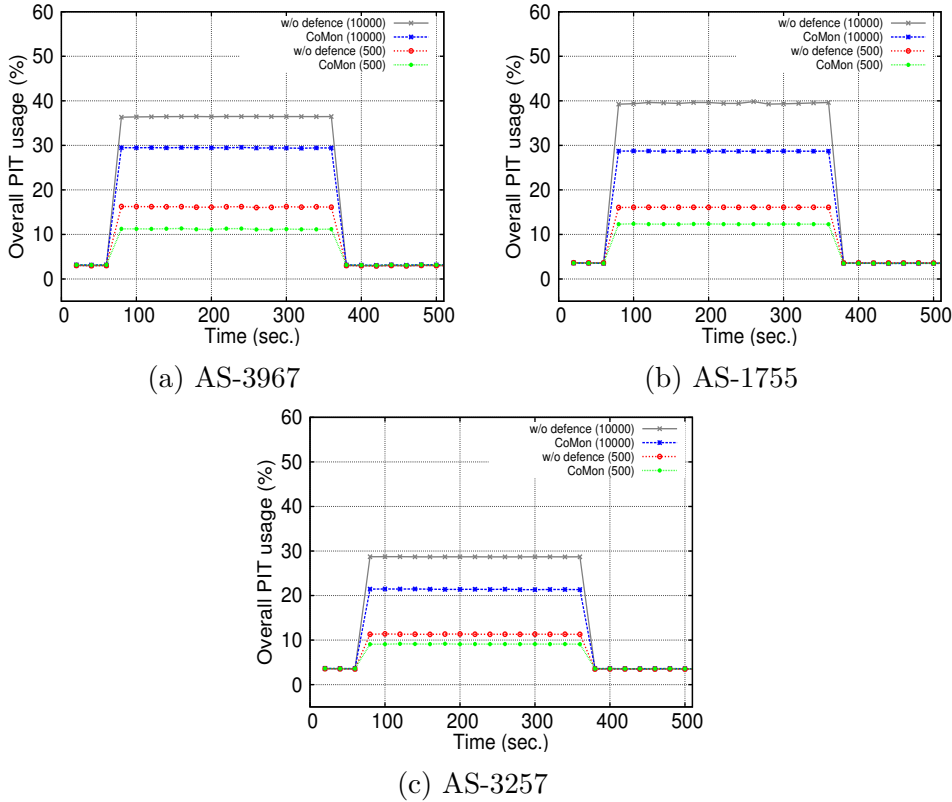


Figure 8.4: Global PIT usage under two attack rates (attack period: seconds 61 – 360)

Note that the adversary nodes in our setup issue malicious interest packets through only $0.25 \times 0.7 = 0.175$ of the nodes to target just one content provider (accessible through one out of three gateway nodes). Therefore, it is likely that there are several nodes not locating on the paths which are used by malicious interest packets, and they hence are not affected by the attack. Such nodes, however, are also counted when calculating the global PIT usage. This explains why the improvement values which are reported by the second metric are smaller than those which are reported by the first metric.

In addition to the effectiveness of our mechanism, it is important to note

Table 8.2: Statistics of global PIT usage during attack periods

		Without defence		CoMon	
		Attack rate (ipps)		Attack rate (ipps)	
		500	10000	500	10000
AS-3967	Minimum	13.99	35.93	9.03	26.01
	Maximum	16.79	37.73	12.59	31.96
	Median	15.57	36.08	11.07	29.13
	Average	15.54	36.01	11.04	28.97
	Std. dev.	0.87	1.13	1.09	1.67
AS-1755	Minimum	14.15	35.32	10.50	26.15
	Maximum	17.49	42.69	13.44	34.52
	Median	16.06	39.21	12.25	30.39
	Average	15.99	39.04	12.23	30.63
	Std. dev.	1.11	2.09	0.93	2.62
AS-3257	Minimum	8.89	25.01	8.52	19.70
	Maximum	14.01	33.99	12.98	23.37
	Median	11.35	29.36	10.59	21.47
	Average	11.51	29.48	10.69	21.49
	Std. dev.	1.42	2.60	1.38	1.07

its high responsiveness: the results in Figure 8.3 and Figure 8.4 show that the reaction against potential IFAs start very shortly after the beginning of the attack, and it also stops very shortly after the end of the attack. This indicates that our detection algorithm results in very few false positives. That is to say, the defence mechanism rarely harms legitimate packets even while the network is free of attacks.

Signalling Overhead

Last but not least, we plot the signalling overhead that is incurred by our defence mechanism during an exemplary simulation run in Figure 8.5. We also summarize the main related statistics of 20 simulation runs in Table 8.3. We can see that the overhead is very marginal both when no attack exists as well as during the attack period. We also can see that the signalling overhead is higher during the attack period, and it increases with the attack rate. This is because the amount of information that is exchanged between the AC and the MNs (*e.g.* observed content names and their information) also increases with the attack rate. Note also that the signalling overhead reported here is even smaller than the overhead

that is reported in Figure 7.9. This is mainly because coordination in the case of defending against IFAs does not involve sending information from the AC to the NNs⁴. In addition, the amount of information exchanged between the AC and the MNs is larger in the cache coordination case.

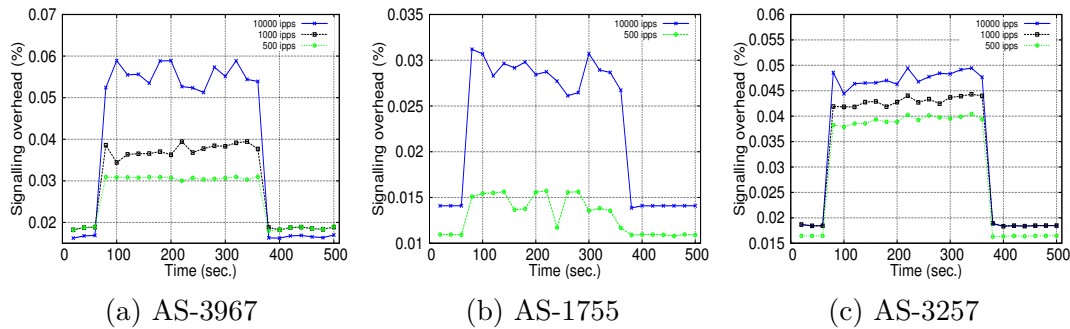


Figure 8.5: Signalling overhead under two attack rates (attack period: seconds 61 – 360)

Table 8.3: Statistics of signalling overhead (percentage) during attack periods

		Attack rate (ipps)	
		500	10000
AS-3967	Minimum	0.03001	0.05126
	Maximum	0.03098	0.05890
	Median	0.03050	0.05511
	Average	0.03050	0.05522
	Std. dev.	0.00028	0.00223
AS-1755	Minimum	0.0116	0.02612
	Maximum	0.0157	0.03120
	Median	0.0137	0.02846
	Average	0.0137	0.02865
	Std. dev.	0.0012	0.00144
AS-3257	Minimum	0.03789	0.04440
	Maximum	0.04043	0.04945
	Median	0.03945	0.04813
	Average	0.03944	0.04808
	Std. dev.	0.00017	0.00041

⁴ In the cache coordination case, the AC sends each NN the corresponding cache assignments.

8.6 Summary

We presented in this chapter a new defence mechanism for the so-called Interest Flooding Attack in NDN. The attack detection is basically performed by a small number of monitoring nodes, from which a network-wide knowledge of traffic and forwarding states can be acquired, with the aid of a centralized controller. Reactions against potential attacks are also performed by the monitoring nodes.

Table 8.4 summarizes the design elements and features that enable our solution to fulfil its preset design requirements (Section 8.3.1).

Table 8.4: CoMon (coordinated defence): Mapping design elements to the requirements

	R1	R2	R3	R4	R5
The AC has network-wide information and shares it with the MNs	✓				
MAR and FTBM enable for full traffic coverage	✓				
Placement of the MNs near clients		✓			
Only first MN reacts to potential attacks (per packet)		✓	✓		
Each packet is checked only once			✓		✓
Detection per name-prefix				✓	
Satisfied interests are always accepted				✓	
Monitoring and main defence tasks are performed by a small number of the MNs					✓
No explicit coordination among the nodes					✓

Through extensive simulations, we showed that the proposed mechanism incurs negligible signalling overhead, and that it is highly effective against IFAs: with our defence mechanism, the large majority of legitimate interest packets were satisfied under a low attack rate. Under a massive attack rate, the achieved satisfaction ratio was above intermediate (nearly 65% – 70%). In all experiments, our defence mechanism significantly outperformed a former solution which was considered effective.

The effectiveness of our defence mechanism can be improved (particularly under massive attack rates) by enforcing clients to lower their interest packet rates when the observed attack rate exceeds a certain threshold. We leave the design details and evaluation of this extension for future work.

Summary and Outlook

In this chapter, we first conclude our work with a summary of the presented contributions and results. Finally, we suggest possibilities for future work.

9.1 Summary

We dealt in this dissertation with two systems: (i) Kademlia, the most popular P2P system today, and (ii) NDN, a potential architecture for the future Internet. Both systems aim to achieve efficient content distribution over the Internet. We structured the dissertation accordingly into two parts. We summarize Part 1 and Part 2 in Section 9.1.1 and Section 9.1.2, respectively.

9.1.1 Understanding and Improving Kademlia

We addressed in the first part the problem of characterizing topological and routing table properties of Kademlia. This was motivated by the importance of understanding these properties for assessing and improving the system's performance and robustness. Accurate characterization of these properties can be derived from analysis of graph snapshots captured from a real Kademlia-type system. Capturing such snapshots, however, is challenging because it involves collection of large amount of information, within a short period of time, from large and rapidly changing systems.

We addressed the aforementioned challenge by KadSpider, our own crawler for KAD (an implementation of Kademlia with about four million nodes, of which about half a million nodes can be found online at the same time). We showed that KadSpider, mainly by exploiting KAD design, downloads a complete routing table up to eight times faster (with eight times fewer crawling messages) than prior KAD crawlers. During the time that KadSpider took to capture a graph snapshot (about 908 seconds), almost 96% of the discovered online nodes and 90% of their

routing table entries remained intact. That is to say, the impact of the crawl time on the consistency of the collected information is marginal. In addition, we reported an average snapshot completeness of about 85%. Although this means that the captured snapshots are not identical to the real system graph, both the system size and the properties of interest are maintained in the captured snapshots. We consequently considered the captured snapshots valid for our analysis.

For comparison purposes, we generated synthetic KAD graphs using a novel simulation model of KAD as well as standard graph models (used in prior work to approximate topological properties of Kademlia-type systems). With the goal to make it behaving like the real system, we implemented KAD in our simulation model exactly as it is implemented in eMule (the most popular client software of KAD). We also applied widely accepted churn settings [BH12].

Analysing the measured and synthetic graphs, we showed that the shape of degree distributions and the resilience of graph connectivity to random departures are well modelled by simulations. However, the system size in addition to the ratios of stale routing information and manipulated clients affected the results for certain properties. Most interestingly, the impact of stale routing information was significant: with nearly 88% stale routing table entries in the measured graphs, the complete graph (including active nodes and stale routing table entries) was much more vulnerable to targeted attacks than in the synthetic graphs.

Using real KAD routing tables, we also analysed two previously disregarded routing performance factors: (i) the completeness of routing tables with respect to global knowledge of active identifiers and (ii) the diversity of contacts over the corresponding identifier space sections.

The analysis of the first factor showed that nodes in KAD on average can double the number of active contacts contained in their routing tables, if they would have global system knowledge. Increasing the number of active contacts will improve the routing performance in terms of lookup's hop count and latency as well as the success rate of lookup messages.

Regarding the second factor, it is measured for each routing table bucket independently. We showed that routing table buckets in KAD achieved on average almost half the maximal diversity degree. We also showed that maximizing the diversity can improve the standard routing performance. We consequently proposed and implemented a new neighbour selection scheme in which the nodes aim at maximizing the diversity. The scheme is compatible with Kademlia and its derivatives, and it incurs almost no extra overhead. Small-scale simulations for three notable Kademlia-type systems without churn, in agreement with the derivations of our model for Kademlia routing [RSS15], showed that our scheme enables to achieve an average routing hop count reduction up to nearly 7%. Simulations with churn showed that the reported performance gain increases with the churn rate as long as the success rate of request packets is not very low. In

addition, modified clients, while connecting to the real system, achieved an average improvement of about 4%. The reported improvement values, although small, confirms the utility of our scheme. We also argued that this utility could be higher if the scheme was implemented in a real system completely (*i.e.* in all the nodes). However, evaluating our scheme in such a scenario is technically infeasible, since none of the existing Kademlia-type systems implement our scheme, nor we could enforce all system nodes to use it.

9.1.2 Improving Named-Data Networking

We addressed in the second part of the dissertation two problems in NDN: (i) coordinated cache management and (ii) defending against the so-called Interest Flooding Attack (IFA), an NDN-tailored DDoS attack. We argued that both problems can be effectively addressed in a coordinated way based on timely knowledge of system-wide information. However, realizing such a solution in large networks such as the Internet or Autonomous Systems (ASs) with low overhead is challenging and was not achieved earlier.

Towards this end, we developed CoMon, a framework for AS-wide coordination in NDN. CoMon relies on few nodes to perform monitoring and re-routing of content requests towards in-network caches. These nodes are selected such that they intercept the entire network traffic in an early stage (*i.e.* close to the clients) and coordinate via a centralized controller.

The coordination in the cache management case is based on AS-wide knowledge of both content access information and cache states. Extensive simulations showed that incorporating CoMon in NDN enables to achieve a remarkable caching efficiency gain, compared both to the vanilla NDN as well as to notable cache coordination solutions [LCS06, PCP12]. Most notably, under a bad case of content popularity (MZipf content distribution with $\alpha = 0.8$ and $q > 0$), CoMon outperformed LCD [LCS06] (which in turn outperformed ProbeCache [PCP12] and the vanilla NDN) by about 30% in terms of the fraction of content requests that cross the AS boundaries. Furthermore, CoMon's signalling overhead of coordination was negligible in all conducted experiments.

Finally, motivated by the importance of treating threats in a prospective future Internet architecture, we adapted CoMon to defend against IFAs. The proposed defence mechanism detects and reacts to IFAs based on AS-wide attack-related information close to the potential attack sources. The mechanism assigns attack detection and reaction tasks to the monitoring nodes, in collaboration with the aforementioned controller. We confirmed the effectiveness and feasibility of our defence mechanism via a simulative study. The results showed that the mechanism offers a highly effective defence against IFAs, outperforming both the original NDN system and the satisfaction-based pushback (SBP) mechanism [AMM⁺13] (which

was considered effective), causing negligible signalling overhead. In particular, our mechanism increased the satisfaction ratio of legitimate interest packets by about 9% – 37% under a relatively low-rate attack, and by about 27% – 42% under a massive-rate attack. In contrast, SBP improved the aforementioned ratio under the same low-rate attack by about 2% – 28% only. Even worse, the influence of SBP under the same massive-rate attack was negative.

9.2 Outlook

There are several directions to build on the work that we presented in this dissertation. For instance, we believe that our findings in Chapter 4 about the disparities between simulative results and the actual system should be taken into account when developing simulation models for P2P systems. In addition, designs of existing simulation models should be revised accordingly.

We showed in Chapter 5 that Kademlia-type systems, when implementing our new neighbour selection scheme, can reduce the average lookup’s hop count with negligible overhead. We consequently recommend to use our scheme in future implementations of Kademlia. It also would be interesting to measure the routing performance when our scheme considers not only the diversity of contacts over the respective identifier space sections but also their geographic proximity [Str07, CDHR02, KLKP08]. The same applies for combining our scheme with other routing performance improvements such as [LSMK05]. Since our scheme is compatible with previous improvements, it is expected to enhance their original performance gains further, when it is integrated with them.

We explained in Chapter 7 how CoMon is designed in a generic way so that it can adapt different caching strategies. So far we evaluated CoMon with an exemplary caching strategy: selecting the most frequently requested contents, as observed over the entire AS, for caching. Alternatively, the caching strategy can take other factors into account. For instance, it could be beneficial for the overall caching efficiency to consider the content type and/or the cost of fetching the content from outside the AS.

From the design’s point of view, CoMon can be improved by redesigning and reimplementing the AS controller in a distributed way, both for load balancing and for fault tolerance. Another logical extension is to implement CoMon on a multi-AS scale. In this case, different ASs maintain content-level peering agreements to leverage each other’s cached contents and/or perform inter-AS collaborative defence against IFAs. Such a design, however, creates severe privacy threats as ASs have to reveal sensitive information such as names and access statistics of contents requested by their users. Therefore, a solution for preserving the privacy of published data is required. The challenge in this case is to balance

the trade-off between data privacy and utility of published data for analysis. While several solutions were proposed in the last years to address such a challenge, there are several issues remain to be addressed (for instance, see [GDLS14, RY15]).

Our vision of CoMon is broader than just a framework for coordinating caching-related decisions and for defending against IFAs. In NDN (or ICN in general), CoMon also can be used as an infrastructure for collecting network statistics. It can be also adapted (without substantial changes) to defend against cache pollution attacks. In this case, a new defence mechanism can benefit from the aggregated (or even network-wide) information that is available with CoMon to improve on former defence mechanisms such as CacheShield [XWW12] and [CGT13].

Last but not least, apart from ICN, the main idea of CoMon – relying on a small fraction of well positioned nodes to monitor and coordinate via a centralized controller – can help to solve several monitoring problems in today’s Internet. In particular, accurate online monitoring is essential for determining traffic properties as well as for detecting anomalies and attacks in the Internet. However, with today’s high bandwidth links and consequently massive traffic volumes, online monitoring is hard to perform. With CoMon, the monitoring nodes can be used to collect and reprocess detailed flow records (without repetitions) and send the output to the controller. The controller, in turn, aggregates the information that it receives from the monitoring nodes and after that commands the monitoring nodes to perform proper actions (*e.g.* dropping part of the traffic coming from malicious sources).

Bibliography

- [AHZ15] Eslam AbdAllah, Hossam S Hassanein, and Mohammad Zulkernine. A Survey of Security Attacks in Information-Centric Networking. *IEEE Communications Surveys & Tutorials*, 2015.
- [AMM⁺13] Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang. Interest flooding attack and countermeasures in Named Data Networking. In *IEEE IFIP Networking*, 2013.
- [AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. Network flows: Theory, algorithms, and applications. *Prentice Hall*, 1993.
- [AMZ⁺12] Alexander Afanasyev, Ilya Moiseenko, Lixia Zhang, et al. ndnSIM: NDN simulator for NS-3. *University of California, Los Angeles, Tech. Rep*, 2012.
- [BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 1999.
- [BH12] Ingmar Baumgart and Bernhard Heep. Fast but economical: A simulative comparison of structured peer-to-peer systems. In *IEEE NGI*, 2012.
- [BHK07] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A flexible overlay network simulation framework. In *IEEE Global Internet Symposium*, 2007.
- [BKST13] Chadi Barakat, Abhishek Kalla, Damien Saucez, and Thierry Turletti. Minimizing bandwidth on peering links with deflection in named data networking. In *ICCIT*, 2013.
- [Bru06] Rene Brunner. A Performance Evaluation of the Kad Protocol. *Master thesis, Institut Eurecom and University of Mannheim*, 2006.

- [bt] Bittorrent. <http://www.bittorrent.com/>, accessed: 01.09.2015.
- [BW02] Mudashiru Busari and Carey Williamson. ProWGen: a synthetic workload generation tool for simulation evaluation of web proxy caches. *Elsevier Computer Networks*, 2002.
- [CCGT13] Alberto Compagno, Marco Conti, Paolo Gasti, and Gene Tsudik. Poseidon: Mitigating interest flooding DDoS attacks in named data networking. In *IEEE LCN*, 2013.
- [CDHR02] Miguel Castro, Peter Druschel, Y Charlie Hu, and Antony Rowstron. Exploiting network proximity in distributed hash tables. In *FuDiCo*, 2002.
- [CEBAH00] Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Resilience of the Internet to random breakdowns. *Physical Review Letters*, 2000.
- [CEBAH01] Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Breakdown of the Internet under intentional attack. *Physical Review Letters*, 2001.
- [CGT13] Mauro Conti, Paolo Gasti, and Marco Teoli. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks*, 2013.
- [CHPP12] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. Cache "less for more" in information-centric networks. In *Springer NETWORKING*. 2012.
- [cis14] Cisco Visual Networking Index: Forecast and Methodology, 2013–2018. *CISCO white paper*, 2014.
- [CKR⁺07] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *ACM IMC*, 2007.
- [CMSS13] Damiano Carra, Pietro Michiardi, Hani Salah, and Thorsten Strufe. On the Impact of Incentives in eMule: Analysis and Measurements of a Popular File-Sharing Application. *IEEE JSAC*, 2013.
- [CRR13] Raffaele Chiocchetti, Dario Rossi, and Giuseppe Rossini. ccnSim: An highly scalable CCN simulator. In *IEEE ICC*, 2013.

- [CW07] Scott Crosby and Dan Wallach. An analysis of bittorrent's two kademlia-based DHTs. Technical report, TR07-04, Rice University, 2007.
- [Dan09] Christian Dannewitz. NetInf: An information-centric design for the future internet. In *KuVS Workshop on the Future Internet*, 2009.
- [DGLL07] David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee. A taxonomy of botnet structures. In *IEEE ACSAC*, 2007.
- [DNF⁺08] Carlton R Davis, Stephen Neville, José M Fernandez, Jean-Marc Robert, and John Mchugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *Springer Computer Security-ESORICS*. 2008.
- [DPM11] Steven DiBenedetto, Christos Papadopoulos, and Dan Massey. Routing policies in named data networking. In *SIGCOMM ICN workshop*, 2011.
- [DWFL13] Huichen Dai, Yi Wang, Jindou Fan, and Bin Liu. Mitigate DDoS Attacks in NDN by Interest Traceback. In *INFOCOM Workshops*, 2013.
- [EFGK03] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 2003.
- [EFK13] Gil Einziger, Roy Friedman, and Eyal Kibbar. Kaleidoscope: Adding Colors to Kademlia. In *IEEE P2P*, 2013.
- [emu] The eMule Project. <http://www.emule-project.net/>, accessed: 01.09.2015.
- [ENM⁺12] Suyong Eum, Kiyohide Nakauchi, Masayuki Murata, Yozo Shoji, and Nozomu Nishinaga. CATT: potential based routing with content caching for ICN. In *ICN workshop*, 2012.
- [ER59] Paul Erdős and Alfréd Rényi. On random graphs I. *Publ. Math. Debrecen*, 1959.
- [fac] Facebook. <http://www.facebook.com/>, accessed: 01.09.2015.
- [fia] NSF Future Internet Architecture Project. <http://www.nets-fia.net/>, accessed: 01.09.2015.
- [fli] Flickr. <https://www.flickr.com/>, accessed: 01.09.2015.

- [FPJ⁺07] Jarret Falkner, Michael Piatek, John P John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user DHT. In *ACM IMC*, 2007.
- [Fre77] Linton C Freeman. A set of measures of centrality based on betweenness. *JSTOR Sociometry*, 1977.
- [GCFE13] David Goergen, Thibault Cholez, Jérôme François, and Thomas Engel. Security monitoring for content-centric networking. In *Springer data privacy management and autonomous spontaneous security*. 2013.
- [GDLS14] Aris Gkoulalas-Divanis, Grigorios Loukides, and Jimeng Sun. Publishing data from electronic health records while preserving privacy: A survey of algorithms. *Journal of biomedical informatics*, 2014.
- [gla] G-lab. <http://www.german-lab.de/>, accessed: 01.09.2015.
- [goo] Google+. <https://plus.google.com/>, accessed: 01.09.2015.
- [GTUZ13] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. DoS and DDoS in Named Data Networking. In *IEEE ICCCN*, 2013.
- [HB96] John Hawkinson and Tony Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS) - RFC 1930. 1996.
- [Hee10] Bernhard Heep. R/Kademlia: Recursive and topology-aware overlay routing. In *IEEE ATNAC*, 2010.
- [HKLM03] Andreas Heinemann, Jussi Kangasharju, Fernando Lyardet, and Max Mühlhäuser. iclouds—peer-to-peer information sharing in mobile environments. In *Euro-Par*. 2003.
- [HS08] Mohamed Hefeeda and Osama Saleh. Traffic modeling and proportional partial caching for peer-to-peer systems. *IEEE/ACM Transactions on Networking*, 2008.
- [HSD⁺08] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix C Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. *LEET*, 2008.
- [HYEN09] Duc T Ha, Guanhua Yan, Stephan Eidenbenz, and Hung Q Ngo. On the effectiveness of structural detection and defense against P2P-based botnets. In *IEEE/IFIP DSN*, 2009.

- [Ind14] Cisco Visual Networking Index. The zettabyte era—trends and analysis. *Cisco white paper*, 2014.
- [JAB01] Mihajlo Jovanović, Fred Annexstein, and Kenneth Berman. Modeling peer-to-peer network topologies through small-world models and power laws. In *TELFOR*, 2001.
- [JADH10] Konrad Junemann, Philipp Andelfinger, Jochen Dinger, and Hannes Hartenstein. BitMON: A Tool for Automated Monitoring of the BitTorrent DHT. In *IEEE P2P*, 2010.
- [Jim13] Raúl Jimenez. Distributed Peer Discovery in Large-Scale P2P Streaming Systems: Addressing Practical Problems of P2P Deployments on the Open Internet. *PhD Thesis, KTH*, 2013.
- [JOK11] Raul Jimenez, Flutra Osmani, and Björn Knutsson. Sub-second lookups on a large-scale Kademlia-based overlay. In *IEEE P2P*, 2011.
- [JST⁺09] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *CoNEXT*, 2009.
- [KCC⁺07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM CCR*, 2007.
- [KCTHK09] Hun J Kang, Eric Chan-Tin, Nicholas J Hopper, and Yongdae Kim. Why Kad lookup fails. In *IEEE P2P*, 2009.
- [KF05] Kendy Kutzner and Thomas Fuhrmann. Measuring Large Overlay Networks: The Overnet Example. In *KiVS*, 2005.
- [KLKP08] Sebastian Kaune, Tobias Lauinger, Aleksandra Kovačević, and Konstantin Pussep. Embracing the peer next door: Proximity in kademlia. In *IEEE P2P*, 2008.
- [KLS05] Murali Kodialam, T. V. Lakshman, and Sudipta Sengupta. Configuring networks with content filtering nodes with applications to network security. In *INFOCOM*, 2005.
- [Lau10] Tobias Lauinger. Security & scalability of content-centric networking. *Master thesis, TU Darmstadt and Eurecom*, 2010.

- [LCS06] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Elsevier Performance Evaluation*, 2006.
- [LKR04] Jian Liang, Rakesh Kumar, and Keith W Ross. The kazaa overlay: A measurement study. In *IEEE annual computer communications workshop*, 2004.
- [LMCC10] Xiangtao Liu, Tao Meng, Kai Cai, and Xueqi Cheng. Rainbow: A Robust and Versatile Measurement Tool for Kademlia-Based DHT Networks. In *PDCAT*, 2010.
- [LS11] Zhe Li and Gwendal Simon. Time-shifted TV in content centric networks: The case for cooperative in-network caching. In *IEEE ICC*, 2011.
- [LSG⁺05] Jinyang Li, Jeremy Stribling, Thomer M Gil, Robert Morris, and M Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *P2P Systems*. 2005.
- [LSMK05] Jinyang Li, Jeremy Stribling, Robert Morris, and M Frans Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Usenix NSDI*, 2005.
- [LSS04] Nikolaos Laoutaris, Sofia Syntila, and Ioannis Stavrakakis. Meta algorithms for hierarchical web caches. In *IEEE International Conference on Performance, Computing, and Communications*, 2004.
- [LVT10] Dmitrij Lagutin, Kari Visala, and Sasu Tarkoma. Publish/Subscribe for Internet: PSIRP Perspective. *Future Internet Assembly*, 2010.
- [LXWZ13] Yanhua Li, Haiyong Xie, Yonggang Wen, and Zhi-Li Zhang. Coordinating in-network caching in content-centric networks: model and analysis. In *ICDCS*, 2013.
- [MIF02] Ripeanu Matei, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella network. *IEEE Internet Computing*, 2002.
- [MJ09] Alberto Montresor and Márk Jelasity. PeerSim: A Scalable P2P Simulator. In *IEEE P2P*, 2009.
- [MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, 2002.
- [MR95] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 1995.

- [MSB⁺15] Rodrigo B Mansilha, Lorenzo Saino, Marinho P Barcellos, Massimo Gallo, Emilio Leonardi, Diego Perino, and Dario Rossi. Hierarchical content stores in high-speed icn routers: Emulation and prototype implementation. In *ACM ICN*, 2015.
- [Mys06] David Mysicka. Reverse Engineering of eMule. *Semester thesis, Swiss Federal Institute of Technology (ETH)*, 2006.
- [NS3] The NS-3 Simulator. <https://www.nsnam.org/>, accessed: 01.09.2015.
- [NST13] Xuan Nam Nguyen, Damien Saucez, and Thierry Turletti. Efficient caching in content-centric networks using openflow. In *INFOCOM Workshops*, 2013.
- [NSW01] Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 2001.
- [OMN] OMNeT++ Simulator. <http://www.omnetpp.org/>, accessed: 01.09.2015.
- [PCP12] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic in-network caching for information-centric networks. In *SIGCOMM ICN workshop*, 2012.
- [PGES05] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Springer P2P Systems IV*. 2005.
- [PV11] Diego Perino and Matteo Varvello. A reality check for content centric networking. In *SIGCOMM ICN workshop*, 2011.
- [RR11] Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). Technical report, Telecom ParisTech, 2011.
- [RSS15] Stefanie Roos, Hani Salah, and Thorsten Strufe. Determining the Hop Count in Kademlia-type Systems. In *IEEE ICCCN*, 2015.
- [RY15] Asmaa Hatem Rashid and Norizan Binti Mohd Yasin. Privacy preserving data publishing: Review. *International Journal of Physical Sciences*, 2015.

- [SCB10] Moritz Steiner, Damiano Carra, and Ernst W Biersack. Evaluating and improving the content access in KAD. *Springer P2P networking and applications*, 2010.
- [SENB07] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. A global view of kad. In *ACM IMC*, 2007.
- [SENB09] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Long Term Study of Peer Behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, 2009.
- [SFT13] Vasilis Sourlas, Paris Flegkas, and Leandros Tassiulas. Cache-aware routing in information-centric networks. In *IEEE IFIP IM*, 2013.
- [SGG01] Stefan Saroiu, P Krishna Gummadi, and Steven D Gribble. Measurement study of peer-to-peer file sharing systems. In *Electronic Imaging 2002*, 2001.
- [SGKT06] Kyoungwon Suh, Yang Guo, Jim Kurose, and Don Towsley. Locating network monitors: complexity, heuristics, and coverage. *Elsevier Computer Communications*, 2006.
- [SGR⁺11] Dominik Stingl, Christian Gross, Julius Rückert, Leonhard Nobach, Aleksandra Kovacevic, and Ralf Steinmetz. PeerfactSim.KOM: A simulation framework for peer-to-peer systems. In *HPCS*, 2011.
- [Sil99] ZK Silagadze. Citations and the Zipf-Mandelbrot’s law. *arXiv preprint physics/9901035*, 1999.
- [SKK08] Guenther Starnberger, Christopher Kruegel, and Engin Kirda. Overbot: a botnet protocol based on Kademlia. In *SecureComm*, 2008.
- [SLYJ13] Sumanta Saha, Andrey Lukyanenko, and Antti Yla-Jaaski. Cooperative Caching through Routing Control in Information-Centric Networks. In *INFOCOM*, 2013.
- [SMW02] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM*, 2002.
- [SR05a] Daniel Stutzbach and Reza Rejaie. Capturing accurate snapshots of the Gnutella network. In *IEEE Global Internet Symposium*, 2005.
- [SR05b] Daniel Stutzbach and Reza Rejaie. Evaluating the accuracy of captured snapshots by peer-to-peer crawlers. In *PAM*. 2005.

- [SR06a] Daniel Stutzbach and Reza Rejaie. Improving Lookup Performance Over a Widely-Deployed DHT. In *INFOCOM*, 2006.
- [SR06b] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *ACM IMC*, 2006.
- [SR09] Daniel Stutzbach and Reza Rejaie. Characterization of P2P Systems. *Handbook of Peer-to-Peer Networking*, 2009.
- [SRS08] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. *IEEE/ACM Transactions on Networking*, 2008.
- [SRS14a] Hani Salah, Stefanie Roos, and Thorsten Strufe. Characterizing Graph-Theoretic Properties of a Large-Scale Overlay: Measurements vs. Simulations. In *IEEE ISCC*, 2014.
- [SRS14b] Hani Salah, Stefanie Roos, and Thorsten Strufe. Diversity entails improvement: A new neighbour selection scheme for Kademlia-type systems. In *IEEE P2P*, 2014.
- [SS13a] Hani Salah and Thorsten Strufe. Capturing connectivity graphs of a large-scale p2p overlay network. In *ICDCS's HotPOST Workshop*, 2013.
- [SS13b] Benjamin Schiller and Thorsten Strufe. GTNA 2.0 - a framework for rapid prototyping and evaluation of routing algorithms. In *SummerSim*, 2013.
- [SS15] Hani Salah and Thorsten Strufe. CoMon: An Architecture for Coordinated Caching and Cache-Aware Routing in CCN. In *IEEE CCNC*, 2015.
- [SSS14] Hani Salah, Benjamin Schiller, and Thorsten Strufe. CoMon: A System Architecture for Improving Caching in CCN - (poster paper). In *INFOCOM WKSHPS*, 2014.
- [Str07] Thorsten Strufe. Ein Peer-to-Peer-basierter Ansatz für die Live-Übertragung multimedialer Datenströme. *PhD Dissertation, TU Ilmenau*, 2007.
- [SWS15a] Hani Salah, Julian Wulfheide, and Thorsten Strufe. Coordination Supports Security: A New Defence Mechanism Against Interest Flooding in NDN. In *IEEE LCN*, 2015.

- [SWS15b] Hani Salah, Julian Wulfheide, and Thorsten Strufe. Lightweight Coordinated Defence Against Interest Flooding Attacks in NDN - (poster paper). In *INFOCOM WKSHPS*, 2015.
- [TBF⁺03] Wesley W Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro P Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *ACM DEBS*, 2003.
- [ut] uTorrent Client. <http://www.utorrent.com/>, accessed: 01.09.2015.
- [vuz] Vuze Client. <http://www.vuze.com/>, accessed: 01.09.2015.
- [WBK14] Liang Wang, Suzan Bayhan, and Jussi Kangasharju. Effects of Cooperation Policy and Network Topology on Performance of In-Network Caching. *IEEE Communications Letters*, 2014.
- [WC11] Qi Wu and Xingshu Chen. Advanced Distributed Crawling System for the KAD Network. *Journal of Computational Information Systems*, 2011.
- [Wid12] Indra Widjaja. Towards a flexible resource management system for content centric networking. In *IEEE ICC*, 2012.
- [WK13] Liang Wang and Jussi Kangasharju. Measuring large-scale distributed systems: case of BitTorrent Mainline DHT. In *IEEE P2P*, 2013.
- [WLV⁺12] Yaogong Wang, Kyunghan Lee, Balakrishna Venkataraman, Ravi L Shamanna, Injong Rhee, and Sunhee Yang. Advertising cached contents in the control plane: Necessity and feasibility. In *INFOCOM Workshops*, 2012.
- [WTCT⁺08] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. Attacking the Kad network. In *SecureComm*, 2008.
- [WXLZ06] Chen Wang, Li Xiao, Yunhao Liu, and Pei Zheng. DiCAS: An efficient distributed caching mechanism for P2P systems. *TPDS*, 2006.
- [WZB13] Jason Min Wang, Jun Zhang, and Brahim Bensaou. Intra-AS cooperative caching for content-centric networks. In *SIGCOMM ICN workshop*, 2013.

- [WZQZ14] Kai Wang, Huachun Zhou, Yajuan Qin, and Hongke Zhang. Cooperative-Filter: countering Interest flooding attacks in named data networking. *Springer Soft Computing*, 2014.
- [XVS⁺13] George Xylomenos, Christopher N Ververidis, Vasilios Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, George C Polyzos, et al. A survey of information-centric networking research. *IEEE Communication Magazine*, 2013.
- [XWW12] Mengjun Xie, Indra Widjaja, and Haining Wang. Enhancing cache robustness for content-centric networking. In *INFOCOM*, 2012.
- [YFX⁺09] Jie Yu, Chengfang Fang, Jia Xu, Ee-Chien Chang, and Zhoujun Li. ID repetition in Kad. In *IEEE P2P*, 2009.
- [YLX⁺11] Jie Yu, Liming Lu, Peng Xiao, Zhoujun Li, and Yuan Zhou. Monitoring, analyzing and characterizing lookup traffic in a large-scale DHT. *Computer Communications*, 2011.
- [you] YouTube. <http://www.youtube.com/>, accessed: 01.09.2015.
- [ZLL13] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: a survey. *Elsevier Computer Networks*, 2013.

Appendix A

Search Keywords Used in KAD Measurements

Table A.1: Search keywords and prefixes of their MD4 hashes (selected from [SCB10])

00 ice	09 cosmic	13 park	1c lingerie	26 queen
00 girls	0a feathers	13 carlisle	1d singer	26 monster
01 things	0a cage	14 toms	1d americas	27 cool
01 mess	0b henrys	14 cinema	1e jitters	27 financially
02 analog	0b trouble	15 days	1e interception	28 racing
02 tons	0c contraband	15 licking	1f boy	28 olympics
03 volume	0c terrible	16 jubilee	1f well	29 north
03 daze	0d hats	16 melody	20 sacrifice	29 wooden
04 pup	0d came	17 timeshift	20 escape	2a financing
04 loving	0e true	17 wart	21 timebomb	2a charlie
05 chicks	0e cave	18 backstage	21 thousand	2b gottfried
05 megaforce	0f old	18 stielow	22 tabby	2b luck
06 babes	0f fine	19 paulina	22 yesterdays	2c bei
06 verdict	10 megan	19 questions	23 trial	2c soldiers
07 mellos	10 behind	1a behave	23 daddy	2d romantic
07 beyond	11 did	1a fast	24 island	2d chicken
08 melodious	11 mine	1b believer	24 finest	2d legal
08 diner	12 bela	1b anonymous	25 voices	2e rolling
09 here	12 druggists	1c venture	25 guys	2f finale

2f killing	40 sayin	50 your	61 cut	71 helens
30 people	41 nutty	51 yellowstone	61 sombras	72 rainbow
30 one	42 xix	51 citizen	62 stroke	72 smooth
31 summer	42 tinnitus	52 tunnels	62 struggles	73 years
31 witch	42 dont	52 future	63 presents	75 goon
32 timemaster	42 muscle	53 are	63 tushyland	75 dogumentary
32 christa	43 yentl	53 godiva	64 yearbook	75 age
33 cab	43 yesyears	54 ghost	64 yellowneck	75 welcome
33 wife	44 yeti	54 particular	65 presidential	75 beijing
34 gobblers	44 gone	55 belladonna	65 parade	76 wrapped
34 yellowcard	45 guide	55 skelter	66 step	76 trail
35 meine	45 bears	56 stalker	66 bondage	77 solitary
36 favorite	46 great	56 dogs	67 touching	77 women
36 fever	46 kid	57 beer	67 machine	78 einstein
36 pierces	47 scarecrow	57 megadeth	68 punkins	78 dot
37 lickin	47 etiquette	58 hour	68 thru	79 moments
37 songs	48 gold	58 deluxe	69 hoppy	79 plays
38 regiment	48 timeless	59 gang	69 bell	7a hand
38 lariat	49 naval	59 tommy	6a solutions	7a megiddo
39 scott	49 baffles	5a bellamy	6a megalodon	7b soldier
39 eye	4a dicks	5a claudine	6b ever	7b who
3a has	4a bellas	5b its	6b prohibited	7c hop
3a bel	4b brown	5b come	6c dancing	7c youth
3b timesaver	4b kahan	5c boyd	6c buff	7d melodies
3b timequest	4c eyes	5c police	6d magnum	7d highway
3c international	4c russia	5d crooks	6d leagues	7e mates
3c chickens	4d heroes	5d dust	6e powder	7e lean
3d solitudes	4d standard	5e ram	6e hermann	7f doctor
3d settlement	4e some	5e swordfish	6f refinement	7f hunters
3e orgasm	4e tower	5f chaser	6f then	80 bryon
3e seven	4f strong	5f wally	70 geniuses	80 solve
3f art	4f justice	60 makes	70 follies	81 percent
3f blast	50 find	60 typhoon	71 chocolate	81 gay

82 badge	92 paul	a3 path	b3 believe	c4 family
82 genius	93 yearling	a3 revue	b4 betty	c4 grace
83 nurse	93 skin	a4 trackers	b4 timothy	c5 melodrama
83 wind	94 contortionist	a4 swallowed	b5 eleazar	c5 eleven
84 bettys	95 being	a5 american	b5 mejor	c6 square
84 private	95 solitaire	a5 cupid	b6 tinas	c6 table
85 reward	95 toy	a6 timing	b7 voyage	c7 story
85 door	96 shriner	a6 man	b7 doings	c7 prints
86 too	96 timothys	a7 bridges	b7 lift	c8 sisters
86 nate	97 etheridge	a7 crevasse	b8 lights	c8 double
87 million	97 taker	a8 point	b8 sombrerhos	c9 rock
87 party	98 collectors	a8 wolf	b9 soloist	c9 take
88 train	98 timesweep	a9 making	ba fantasies	ca romeo
88 touch	99 lily	a9 divine	ba symphony	ca dream
89 tail	99 sus	aa solomon	ba safe	cb mein
89 grey	9a boyfriend	aa rare	bb trans	cb and
8a mekong	9a unleashed	ab show	bc tina	cc spy
8a nickleby	9b they	ab xmas	bc hoss	cc lane
8b romance	9b summers	ac bellclair	bc edge	cd bath
8b gila	9c miss	ac wallpaper	bd isle	cd cup
8c freaks	9c anatomy	ad society	bd ozzy	ce men
8c screen	9d mazie	ad warrior	be air	ce dobbs
8d chile	9d yes	ae strutters	be hero	cf awards
8d have	9e midnight	ae beethoven	bf picnic	cf from
8e belle	9e times	af timecode	bf home	d0 solution
8e believers	9f crime	af blues	c0 bronc	d0 valentine
8f help	9f streak	b0 wounds	c0 victory	d1 mama
8f date	a0 admiral	b0 food	c1 timepiece	d1 exposure
90 confession	a1 damnd	b1 timmys	c1 mitchell	d2 ordinary
90 print	a1 maine	b1 meeting	22 stage	d2 yee
91 nightfire	a2 pours	b2 detective	c2 bubbly	d3 excursionists
91 obsession	a2 executives	b2 desires	c3 fire	d3 roller
92 bellboy	a2 vengeance	b3 folks	c3 blackhand	d4 fingered

d4 bridegroom	dd killer	e6 madame	ef beimo	f9 timelock
d5 pigs	de melinda	e7 proposito	f0 went	f9 solitaires
d5 quack	de jack	e7 you	f0 belated	f9 something
d6 smitty	df carter	e8 pawn	f1 vendetta	fa triangle
d6 wish	df finally	e8 lick	f1 sex	fa vincent
d7 mel	e0 meet	e9 news	f2 finiculee	fb hammer
d7 turns	e0 clits	e9 feeding	f2 drink	fb melanzane
d8 number	e1 liar	ea cheers	f3 barn	fc fat
d8 nudity	e1 rescue	ea manhandled	f3 two	fc magdalene
d9 call	e2 body	eb forever	f4 crimes	fd mother
d9 elopement	e2 daughter	ec adversary	f4 sombra	fd panic
da mix	e3 taboo	ec tinkercrank	f5 kal	fe bernsteins
da somay	e3 financial	ec star	f5 belly	fe vision
db donald	e4 cumpany	ed republic	f6 habaneras	ff amateur
db for	e4 elephant	ed michigan	f6 dynamite	ff picture
dc belinda	e5 finicula	ee pride	f7 timecollapse	
dc psychotic	e5 cruise	ee self	f7 pan	
dd consents	e6 solus	ef sinema	f8 steamer	

Acronyms

AS	Autonomous System
AC	AS Controller
BA	Barabási-Albert
BC	Betweenness Centrality
CAR	Cache-Aware Routing
CCN	Content-Centric Networking
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CI	Confidence Interval
CS	Cache Store
(D)DoS	(Distributed) Denial-of-Service
DHT	Distributed Hash Table
DONA	Data Oriented Network Architecture
ER	Erdős-Rényi
FIA	Future Internet Architecture
FIB	Forwarding Information Base
FoC	Feasibility of Changing
FTBM	Forward-Till-Be-Monitored
GBC	Group Betweenness Centrality
GTNA	Graph-Theoretic Network Analyzer
ICN	Information-Centric Networking
IFA	Interest Flooding Attack
LCD	Leave Copy Down
LFU	Least Frequently Used
LRU	Least Recently Used
MAR	Monitor-Aware Routing
MCD	Move Copy Down
MDHT	Mainline DHT

MN	Monitoring Node
MZipf	Mandelbrot-Zipf
NAT	Network Address Translation
NDN	Named-Data Networking
NetInf	Network of Information
NN	NDN Node
NSF	National Science Foundation
P2P	Peer-to-Peer
PIT	Pending Interest Table
PRCS	Placement based on covered Routes and Closeness to Sources
PSIRP	Publish/Subscribe Internet Routing Paradigm
pub/sub	publish/subscribe
QoE	Quality of Experience
RCF	Routing-with-Content-Filtering
RTT	Round Trip Time
SHR	Server Hit Ratio

Author's Publications

Journal article

- Damiano Carra, Pietro Michiardi, **Hani Salah**, and Thorsten Strufe. On the Impact of Incentives in eMule: Analysis and Measurements of a Popular File-Sharing Application. In IEEE JSAC, 2013.

Refereed conference papers

- **Hani Salah**, Julian Wulfheide, and Thorsten Strufe. Coordination Supports Security: A New Defence Mechanism Against Interest Flooding in NDN. In IEEE LCN, 2015.
- Stefanie Roos, **Hani Salah**, and Thorsten Strufe. Determining the Hop Count in Kademlia-type Systems. In IEEE ICCCN, 2015.
- **Hani Salah** and Thorsten Strufe. CoMon: An Architecture for Coordinated Caching and Cache-Aware Routing in CCN. In IEEE CCNC, 2015.
- Thomas Paul, Stephen Stephen, **Hani Salah**, and Thorsten Strufe. The Students' Portal of Ilmenau: A Holistic OSN's User Behaviour Model. In PICCIT, 2015.
- **Hani Salah**, Stefanie Roos, and Thorsten Strufe. Diversity Entails Improvement: A New Neighbour Selection Scheme for Kademlia-type Systems. In IEEE P2P, 2014.
- **Hani Salah**, Stefanie Roos, and Thorsten Strufe. Characterizing Graph-Theoretic Properties of a Large-Scale Overlay: Measurements vs. Simulations. In IEEE ISCC, 2014.¹

¹ Best student paper in IEEE ISCC 2014.

Refereed workshop paper

- **Hani Salah** and Thorsten Strufe. Capturing Connectivity Graphs of a Large-Scale P2P Overlay Network. In ICDCS's HotPOST Workshop, 2013.

Poster papers

- **Hani Salah**, Julian Wulfheide, and Thorsten Strufe. Lightweight Coordinated Defence Against Interest Flooding Attacks in NDN. In INFOCOM WKSHPS, 2015.
- **Hani Salah**, Benjamin Schiller, and Thorsten Strufe. CoMon: A System Architecture for Improving Caching in CCN. In INFOCOM WKSHPS, 2014.

Wissenschaftlicher Werdegang des Verfassers

02/1999–10/2003

Studium (B.Sc.)

Electrical Engineering / Computer Systems Engineering
Palestine Polytechnic University
Hebron, Palästina

11/2003–07/2005

Beschäftigung als Lehrassistent

Palestine Polytechnic University
Hebron, Palästina

08/2005–01/2007

Studium (M.Sc.)

Information Technology / Internetworking
The Royal Institute of Technology (KTH)
Stockholm, Schweden
Masterarbeitsthema: Evaluation for the Current Solutions of
BGP Security

02/2007–09/2010

Beschäftigung als Dozent

Palestine Polytechnic University
Hebron, Palästina
Themen: computer networks, operating systems

4/2011–03/2015

DAAD Stipendiat

Fachgebiet P2P Netzwerke, Fachbereich Informatik

Technische Universität Darmstadt

Darmstadt, Deutschland

Seit 04/2015

Beschäftigung als Wissenschaftlicher Mitarbeiter

Fachgebiet Knowledge Engineering, Fachbereich Informatik

Technische Universität Darmstadt u. flexOptix GmbH

Darmstadt u. Dietzenbach, Deutschland

